Moz Harness Documentation

Release 0.1

aki and a cast of tens!

August 19, 2015

Contents

Contents:

CHAPTER 1

mozharness

1.1 mozharness package

1.1.1 Subpackages

mozharness.base package

Subpackages

mozharness.base.vcs package

Submodules

mozharness.base.vcs.gittool module

class mozharness.base.vcs.gittool.**GittoolParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can find the "Got revision" string from gittool.py output

got_revision = None

got_revision_exp = <_sre.SRE_Pattern object>

parse_single_line(*line*)

class mozharness.base.vcs.gittool.**GittoolVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

mozharness.base.vcs.hgtool module

class mozharness.base.vcs.hgtool.**HgtoolParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can find the "Got revision" string from hgtool.py output

got_revision = None

got_revision_exp = <_sre.SRE_Pattern object>

parse_single_line(*line*)

class mozharness.base.vcs.hgtool.**HgtoolVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

mozharness.base.vcs.mercurial module Mercurial VCS support.

Largely copied/ported from https://hg.mozilla.org/build/tools/file/cf265ea8fb5e/lib/python/util/hg.py .

class mozharness.base.vcs.mercurial.**MercurialVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*, object

apply_and_push(*localrepo*, *remote*, *changer*, *max_attempts=10*, *ssh_username=None*, *ssh_key=None*)

This function calls 'changer' to make changes to the repo, and tries its hardest to get them to the origin repo. 'changer' must be a callable object that receives two arguments: the directory of the local repository, and the attempt number. This function will push ALL changesets missing from remote.

cleanOutgoingRevs(*reponame*, *remote*, *username*, *sshKey*)

clone(*repo*, *dest*, *branch=None*, *revision=None*, *update_dest=True*)

Clones hg repo and places it at *dest*, replacing whatever else is there. The working copy will be empty.

If *revision* is set, only the specified revision and its ancestors will be cloned. If revision is set, branch is ignored.

If *update_dest* is set, then *dest* will be updated to *revision* if set, otherwise to *branch*, otherwise to the head of default.

common_args(*revision=None*, *branch=None*, *ssh_username=None*, *ssh_key=None*)

Fill in common hg arguments, encapsulating logic checks that depend on mercurial versions and provided arguments

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

get_branch_from_path(*path*)

get_branches_from_path(*path*)

get_repo_name(*repo*)

get_repo_path(*repo*)

get_revision_from_path(*path*)

Returns which revision directory *path* currently has checked out.

hg_ver()

Returns the current version of hg, as a tuple of (major, minor, build)

out(*src*, *remote*, ***kwargs*)
Check for outgoing changesets present in a repo

pull(*repo*, *dest*, *update_dest=True*, ***kwargs*)
Pulls changes from hg repo and places it in *dest*.

If *revision* is set, only the specified revision and its ancestors will be pulled.

If *update_dest* is set, then *dest* will be updated to *revision* if set, otherwise to *branch*, otherwise to the head of default.

push(*src*, *remote*, *push_new_branches=True*, ***kwargs*)

query_can_share()

share(*source*, *dest*, *branch=None*, *revision=None*)
Creates a new working directory in "dest" that shares history with "source" using Mercurial's share extension

update(*dest*, *branch=None*, *revision=None*)
Updates working copy *dest* to *branch* or *revision*. If revision is set, branch will be ignored. If neither is set then the working copy will be updated to the latest revision on the current branch. Local changes will be discarded.

mozharness.base.vcs.mercurial.**make_hg_url**(*hg_host*, *repo_path*, *protocol='http'*, *revision=None*, *filename=None*)
Helper function.

Construct a valid hg url from a base hg url (hg.mozilla.org), repo_path, revision and possible filename

mozharness.base.vcs.vcsbase module Generic VCS support.

class mozharness.base.vcs.vcsbase.**MercurialScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSScript*

default_vcs = 'hg'

class mozharness.base.vcs.vcsbase.**VCSMixin**
Bases: object

Basic VCS methods that are vcs-agnostic. The vcs_class handles all the vcs-specific tasks.

query_dest(*kwargs*)

vcs_checkout(*vcs=None*, *error_level='fatal'*, ***kwargs*)
Check out a single repo.

vcs_checkout_repos(*repo_list*, *parent_dir=None*, *tag_override=None*, ***kwargs*)
Check out a list of repos.

class mozharness.base.vcs.vcsbase.**VCSScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSMixin*, *mozharness.base.script.BaseScript*

pull(*repos=None*, *parent_dir=None*)

mozharness.base.vcs.vcssync module Generic VCS support.

class mozharness.base.vcs.vcssync.**VCSSyncScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSScript*

notify(*message=None*, *fatal=False*)
Email people in the notify_config (depending on status and failure_only)

start_time = 1440001632.71014

Module contents

Submodules

mozharness.base.config module

Generic config parsing and dumping, the way I remember it from scripts gone by.

The config should be built from script-level defaults, overlaid by config-file defaults, overlaid by command line options.

> **(For buildbot-analogues that would be factory-level defaults,** builder-level defaults, and build request/scheduler settings.)

The config should then be locked (set to read-only, to prevent runtime alterations). Afterwards we should dump the config to a file that is uploaded with the build, and can be used to debug or replicate the build at a later time.

TODO:

- check_required_settings or something – run at init, assert that these settings are set.

class mozharness.base.config.**BaseConfig**(*config=None*, *initial_config_file=None*, *config_options=None*, *all_actions=None*, *default_actions=None*, *volatile_config=None*, *option_args=None*, *require_config_file=False*, *append_env_variables_from_configs=False*, *usage='usage: %prog [options]'*)

Bases: object

Basic config setting/getting.

get_actions()

get_cfgs_from_files(*all_config_files*, *options*)

Returns the configuration derived from the list of configuration files. The result is represented as a list of *(filename, config_dict)* tuples; they will be combined with keys in later dictionaries taking precedence over earlier.

all_config_files is all files specified with *–config-file* and *–opt-config-file*; *options* is the argparse options object giving access to any other command-line options.

This function is also responsible for downloading any configuration files specified by URL. It uses parse_config_file in this module to parse individual files.

This method can be overridden in a subclass to add extra logic to the way that self.config is made up. See *mozharness.mozilla.building.buildbase.BuildingConfig* for an example.

get_read_only_config()

list_actions()

parse_args(*args=None*)

Parse command line arguments in a generic way. Return the parser object after adding the basic options, so child objects can manipulate it.

set_config(*config*, *overwrite=False*)

This is probably doable some other way.

verify_actions(*action_list*, *quiet=False*)

verify_actions_order(*action_list*)

class mozharness.base.config.**ExtendOption**(**opts*, ***attrs*)
Bases: optparse.Option

from http://docs.python.org/library/optparse.html?highlight=optparse#adding-new-actions

ACTIONS = ('store', 'store_const', 'store_true', 'store_false', 'append', 'append_const', 'count', 'callback', 'help', 'versio

ALWAYS_TYPED_ACTIONS = ('store', 'append', 'extend')

STORE_ACTIONS = ('store', 'store_const', 'store_true', 'store_false', 'append', 'append_const', 'count', 'extend')

TYPED_ACTIONS = ('store', 'append', 'callback', 'extend')

take_action(*action*, *dest*, *opt*, *value*, *values*, *parser*)

class mozharness.base.config.**ExtendedOptionParser**(***kwargs*)
Bases: optparse.OptionParser

OptionParser, but with ExtendOption as the option_class.

class mozharness.base.config.**LockedTuple**
Bases: tuple

class mozharness.base.config.**ReadOnlyDict**(*dictionary*)
Bases: dict

clear(**args*)

lock()

pop(**args*)

popitem(**args*)

setdefault(**args*)

update(**args*)

mozharness.base.config.**download_config_file**(*url*, *file_name*)

mozharness.base.config.**make_immutable**(*item*)

mozharness.base.config.**parse_config_file**(*file_name*, *quiet=False*, *search_path=None*, *config_dict_name='config'*)
Read a config file and return a dictionary.

mozharness.base.errors module

Generic error lists.

Error lists are used to parse output in mozharness.base.log.OutputParser.

Each line of output is matched against each substring or regular expression in the error list. On a match, we determine the 'level' of that line, whether IGNORE, DEBUG, INFO, WARNING, ERROR, CRITICAL, or FATAL.

TODO: Context lines (requires work on the OutputParser side)

TODO: We could also create classes that generate these, but with the appropriate level (please don't die on any errors; please die on any warning; etc.) or platform or language or whatever.

exception mozharness.base.errors.**VCSException**
Bases: exceptions.Exception

mozharness.base.gaia_test module

mozharness.base.log module

Generic logging classes and functionalities for single and multi file logging. Capturing console output and providing general logging functionalities.

Attributes:

FATAL_LEVEL (int): constant logging level value set based on the logging.CRITICAL value

DEBUG (str): mozharness *debug* log name INFO (str): mozharness *info* log name WARNING (str): mozharness *warning* log name CRITICAL (str): mozharness *critical* log name FATAL (str): mozharness *fatal* log name IGNORE (str): mozharness *ignore* log name LOG_LEVELS (dict): mapping of the mozharness log level names to logging values ROOT_LOGGER (logging.Logger): instance of a logging.Logger class

TODO: - network logging support. - log rotation config

class mozharness.base.log.**BaseLogger**(*log_level='info'*, *log_format='%(message)s'*, *log_date_format='%H:%M:%S'*, *log_name='test'*, *log_to_console=True*, *log_dir='.'*, *log_to_raw=False*, *logger_name=''*, *append_to_log=False*)

Bases: object

Base class in charge of logging handling logic such as creating logging files, dirs, attaching to the console output and managing its output.

Attributes: LEVELS (dict): flat copy of the *LOG_LEVELS* attribute of the *log* module.

TODO: status? There may be a status object or status capability in either logging or config that allows you to count the number of error,critical,fatal messages for us to count up at the end (aiming for 0).

LEVELS = {'info': 20, 'warning': 30, 'critical': 50, 'error': 40, 'debug': 10, 'fatal': 60}

add_console_handler(*log_level=None*, *log_format=None*, *date_format=None*)

create a *logging.StreamHandler* using *sys.stderr* for logging the console output and add it to the *all_handlers* member variable

Args:

log_level (str, optional): useless argument. Not used here. Defaults to None.

log_format (str, optional): format used for the Formatter attached to the StreamHandler. Defaults to None.

date_format (str, optional): format used for the Formatter attached to the StreamHandler. Defaults to None.

add_file_handler(*log_path*, *log_level=None*, *log_format=None*, *date_format=None*)

create a *logging.FileHandler* base on the path, log and date format and add it to the *all_handlers* member variable.

Args: log_path (str): filepath to use for the *FileHandler*. log_level (str, optional): useless argument. Not used here.

Defaults to None.

log_format (str, optional): log format to use for the Formatter constructor. Defaults to the current instance log format.

date_format (str, optional): date format to use for the Formatter constructor. Defaults to the current instance date format.

create_log_dir()
create a logging directory if it doesn't exits. If there is a file with same name as the future logging directory it will be deleted.

get_log_formatter(*log_format=None*, *date_format=None*)
create a *logging.Formatter* base on the log and date format.

Args:

log_format (str, optional): log format to use for the Formatter constructor. Defaults to the current instance log format.

date_format (str, optional): date format to use for the Formatter constructor. Defaults to the current instance date format.

Returns: logging.Formatter: instance created base on the passed arguments

get_logger_level(*level=None*)

translate the level name passed to it and return its numeric value according to *LEVELS* values.

Args:

level (str, optional): level name to be translated. Defaults to the current instance *log_level*.

Returns:

int: numeric value of the log level name passed to it or 0 (NOTSET) if the name doesn't exists

init_message(*name=None*)
log an init message stating the name passed to it, the current date and time and, the current working directory.

Args:

name (str, optional): name to use for the init log message. Defaults to the current instance class name.

log_message(*message*, *level='info'*, *exit_code=-1*, *post_fatal_callback=None*)

Generic log method. There should be more options here – do or don't split by line, use os.linesep instead of assuming

, be able to pass in log level by name or number.

Adding the IGNORE special level for runCommand.

Args: message (str): message to log using the current *logger* level (str, optional): log level of the message. Defaults to INFO. exit_code (int, optional): exit code to use in case of a FATAL level is used.

Defaults to -1.

post_fatal_callback (function, optional): function to callback in case of of a fatal log level. Defaults None.

new_logger()
Create a new logger based on the ROOT_LOGGER instance. By default there are no handlers. The new logger becomes a member variable of the current instance as *self.logger*.

class mozharness.base.log.**LogMixin**
Bases: object

This is a mixin for any object to access similar logging functionality

The logging functionality described here is specially useful for those objects with self.config and self.log_obj member variables

critical(*message*)
calls the log method with CRITICAL as logging level

Args: message (str): message to log

debug(*message*)
calls the log method with DEBUG as logging level

Args: message (str): message to log

error(*message*)
calls the log method with ERROR as logging level

Args: message (str): message to log

exception(*message=None*, *level='error'*)
log an exception message base on the log level passed to it.

This function fetches the information of the current exception being handled and adds it to the message argument.

Args:

message (str, optional): message to be printed at the beginning of the log. Default to an empty string.

level (str, optional): log level to use for the logging. Defaults to ERROR

Returns: None

fatal(*message*, *exit_code=-1*)
calls the log method with FATAL as logging level

Args: message (str): message to log exit_code (int, optional): exit code to use for the SystemExit

exception to be raised. Default to -1.

info(*message*)
calls the log method with INFO as logging level

Args: message (str): message to log

log(*message*, *level='info'*, *exit_code=-1*)
log the message passed to it according to level, exit if level == FATAL

Args: message (str): message to be logged level (str, optional): logging level of the message. Defaults to INFO exit_code (int, optional): exit code to log before the scripts calls

SystemExit.

Returns: None

warning(*message*)
calls the log method with WARNING as logging level

Args: message (str): message to log

worst_level(*target_level*, *existing_level*, *levels=None*)
Compare target_level with existing_level according to levels values and return the worst among them.

Args:

target_level (str): minimum logging level to which the current object should be set

existing_level (str): current logging level levels (list(str), optional): list of logging levels names to compare

target_level and existing_level against. Defaults to mozharness log level list sorted from most to less critical.

Returns:

str: the logging lavel that is closest to the first levels value, i.e. levels[0]

class mozharness.base.log.**MultiFileLogger**(*logger_name='Multi'*, *log_format='%(asctime)s %(levelname)8s - %(message)s'*, *log_dir='logs'*, *log_to_raw=True*, ***kwargs*)

Bases: *mozharness.base.log.BaseLogger*

Subclass of the BaseLogger class. Create a log per log level in log_dir. Possibly also output to the terminal and a raw log (no prepending of level or date)

new_logger()

calls the BaseLogger.new_logger method and adds a file handler per logging level in the *LEVELS* class attribute.

class mozharness.base.log.**OutputParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.LogMixin*

Helper object to parse command output.

This will buffer output if needed, so we can go back and mark [(linenum - 10) : linenum+10] as errors if need be, without having to get all the output first.

linenum+10 will be easy; we can set self.num_post_context_lines to 10, and self.num_post_context_lines– as we mark each line to at least error level X.

linenum-10 will be trickier. We'll not only need to save the line itself, but also the level that we've set for that line previously, whether by matching on that line, or by a previous line's context. We should only log that line if all output has ended (self.finish() ?); otherwise store a list of dictionaries in self.context_buffer that is buffered up to self.num_pre_context_lines (set to the largest pre-context-line setting in error_list.)

add_lines(*output*)

process a string or list of strings, decode them to utf-8,strip them of any trailing whitespaces and parse them using *parse_single_line*

strings consisting only of whitespaces are ignored.

Args: output (str | list): string or list of string to parse

parse_single_line(*line*)

parse a console output line and check if it matches one in *error_list*, if so then log it according to *log_output*.

Args: line (str): command line output to parse.

class mozharness.base.log.**SimpleFileLogger**(*log_format='%(asctime)s %(levelname)8s - %(message)s'*, *logger_name='Simple'*, *log_dir='logs'*, ***kwargs*)

Bases: *mozharness.base.log.BaseLogger*

Subclass of the BaseLogger.

Create one logFile. Possibly also output to the terminal and a raw log (no prepending of level or date)

new_logger()

calls the BaseLogger.new_logger method and adds a file handler to it.

mozharness.base.log.**numeric_log_level**(*level*)
Converts a mozharness log level (string) to the corresponding logger level (number). This function makes possible to set the log level in functions that do not inherit from LogMixin

Args: level (str): log level name to convert.

Returns: int: numeric value of the log level name.

mozharness.base.mar module

mozharness.base.parallel module

Generic ways to parallelize jobs.

class mozharness.base.parallel.**ChunkingMixin**
Bases: object

Generic signing helper methods.

query_chunked_list(*possible_list*, *this_chunk*, *total_chunks*, *sort=False*)
Split a list of items into a certain number of chunks and return the subset of that will occur in this chunk.

Ported from build.l10n.getLocalesForChunk in build/tools.

mozharness.base.python module

Python usage, esp. virtualenv.

class mozharness.base.python.**InfluxRecordingMixin**
Bases: object

Provides InfluxDB stat recording to scripts.

This class records stats to an InfluxDB server, if enabled. Stat recording is enabled in a script by inheriting from this class, and adding an influxdb_credentials line to the influx_credentials_file (usually oauth.txt in automation). This line should look something like:

influxdb_credentials = 'http://goldiewilson-onepointtwentyone-1.c.influxdb.com:8086/db/DBNAME/series?u=DBUSERNA

Where DBNAME, DBUSERNAME, and DBPASSWORD correspond to the database name, and user/pw credentials for recording to the database. The stats from mozharness are recorded in the 'mozharness' table.

influxdb_recording_init()

influxdb_recording_post_action(*action*, *success=None*)

influxdb_recording_pre_action(*action*)

record_influx_stat(*json_data*)

record_mach_stats(*action*, *success=None*)

class mozharness.base.python.**ResourceMonitoringMixin**(**args*, ***kwargs*)
Bases: object

Provides resource monitoring capabilities to scripts.

When this class is in the inheritance chain, resource usage stats of the executing script will be recorded.

This class requires the VirtualenvMixin in order to install a package used for recording resource usage.

While we would like to record resource usage for the entirety of a script, since we require an external package, we can only record resource usage after that package is installed (as part of creating the virtualenv). That's just the way things have to be.

class mozharness.base.python.**VirtualenvMixin**(**args*, ***kwargs*)
Bases: object

BaseScript mixin, designed to create and use virtualenvs.

Config items:

- virtualenv_path points to the virtualenv location on disk.
- virtualenv_modules lists the module names.
- MODULE_url list points to the module URLs (optional)

Requires virtualenv to be in PATH. Depends on ScriptMixin

activate_virtualenv()
Import the virtualenv's packages into this Python interpreter.

create_virtualenv(*modules=()*, *requirements=()*)
Create a python virtualenv.

The virtualenv exe can be defined in c['virtualenv'] or c['exes']['virtualenv'], as a string (path) or list (path + arguments).

c['virtualenv_python_dll'] is an optional config item that works around an old windows virtualenv bug.

virtualenv_modules can be a list of module names to install, e.g.

virtualenv_modules = ['module1', 'module2']

or it can be a heterogeneous list of modules names and dicts that define a module by its name, url-or-path, and a list of its global options.

virtualenv_modules = [

{ 'name': 'module1', 'url': None, 'global_options': ['–opt', '–without-gcc']

}, {

'name': 'module2', 'url': 'http://url/to/package', 'global_options': ['–use-clang']

}, {

'name': 'module3', 'url': os.path.join('path', 'to', 'setup_py', 'dir') 'global_options': []

}, 'module4'

]

virtualenv_requirements is an optional list of pip requirements files to use when invoking pip, e.g.,

virtualenv_requirements = ['/path/to/requirements1.txt', '/path/to/requirements2.txt'

]

install_module(*module=None*, *module_url=None*, *install_method=None*, *requirements=()*, *optional=False*, *global_options=[]*, *no_deps=False*, *editable=False*)
Install module via pip.

module_url can be a url to a python package tarball, a path to a directory containing a setup.py (absolute or relative to work_dir) or None, in which case it will default to the module name.

requirements is a list of pip requirements files. If specified, these will be combined with the module_url (if any), like so:

pip install -r requirements1.txt -r requirements2.txt module_url

is_python_package_installed(*package_name*, *error_level='warning'*)
Return whether the package is installed

package_versions(*pip_freeze_output=None*, *error_level='warning'*, *log_output=False*)
reads packages from *pip freeze* output and returns a dict of {package_name: 'version'}

python_paths = {}

query_python_path(*binary='python'*)
Return the path of a binary inside the virtualenv, if c['virtualenv_path'] is set; otherwise return the binary name. Otherwise return None

query_python_site_packages_path()

query_virtualenv_path()

register_virtualenv_module(*name=None*, *url=None*, *method=None*, *requirements=None*, *optional=False*, *two_pass=False*, *editable=False*)
Register a module to be installed with the virtualenv.

This method can be called up until create_virtualenv() to register modules that should be installed in the virtualenv.

See the documentation for install_module for how the arguments are applied.

site_packages_path = None

mozharness.base.script module

Generic script objects.

script.py, along with config.py and log.py, represents the core of mozharness.

class mozharness.base.script.**BaseScript**(*config_options=None*, *ConfigClass=<class 'mozharness.base.config.BaseConfig'>*, *default_log_level='info'*, ***kwargs*)
Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*, object

action_message(*message*)

add_failure(*key*, *message='%(key)s failed.'*, *level='error'*, *increment_return_code=True*)

add_summary(*message*, *level='info'*)

clobber()
Delete the working directory

copy_logs_to_upload_dir()
Copies logs to the upload directory

copy_to_upload_dir(*target*, *dest=None*, *short_desc='unknown'*, *long_desc='unknown'*, *log_level='debug'*, *error_level='error'*, *max_backups=None*, *compress=False*, *upload_dir=None*)
Copy target file to upload_dir/dest.

Potentially update a manifest in the future if we go that route.

Currently only copies a single file; would be nice to allow for recursive copying; that would probably done by creating a helper _copy_file_to_upload_dir().

short_desc and long_desc are placeholders for if/when we add upload_dir manifests.

dump_config(*file_path=None*, *config=None*, *console_output=True*, *exit_on_finish=False*)
Dump self.config to localconfig.json

file_sha512sum(*file_path*)

new_log_obj(*default_log_level='info'*)

query_abs_dirs()
We want to be able to determine where all the important things are. Absolute paths lend themselves well to this, though I wouldn't be surprised if this causes some issues somewhere.

This should be overridden in any script that has additional dirs to query.

The query_* methods tend to set self.VAR variables as their runtime cache.

query_failure(*key*)

return_code

run()
Default run method. This is the "do everything" method, based on actions and all_actions.

First run self.dump_config() if it exists. Second, go through the list of all_actions. If they're in the list of self.actions, try to run self.preflight_ACTION(), self.ACTION(), and self.postflight_ACTION().

Preflight is sanity checking before doing anything time consuming or destructive.

Postflight is quick testing for success after an action.

run_action(*action*)

run_and_exit()
Runs the script and exits the current interpreter.

summarize_success_count(*success_count*, *total_count*, *message='%d of %d successful.'*, *level=None*)

summary()
Print out all the summary lines added via add_summary() throughout the script.

I'd like to revisit how to do this in a prettier fashion.

class mozharness.base.script.**PlatformMixin**
Bases: object

mozharness.base.script.**PostScriptAction**(*action=None*)
Decorator for methods that will be called at the end of each action.

This behaves similarly to PreScriptAction. It varies in that it is called after execution of the action.

The decorated method will receive the action name as a positional argument. It will then receive the following named arguments:

success - Bool indicating whether the action finished successfully.

The decorated method will always be called, even if the action threw an exception.

The return value is ignored.

mozharness.base.script.**PostScriptRun**(*func*)
Decorator for methods that will be called after script execution.

This is similar to PreScriptRun except it is called at the end of execution. The method will always be fired, even if execution fails.

mozharness.base.script.**PreScriptAction**(*action=None*)
Decorator for methods that will be called at the beginning of each action.

Each method on a BaseScript having this decorator will be called during BaseScript.run() before an individual action is executed. The method will receive the action's name as an argument.

If no values are passed to the decorator, it will be applied to every action. If a string is passed, the decorated function will only be called for the action of that name.

The return value of the method is ignored. Exceptions will abort execution.

mozharness.base.script.**PreScriptRun**(*func*)
Decorator for methods that will be called before script execution.

Each method on a BaseScript having this decorator will be called at the beginning of BaseScript.run().

The return value is ignored. Exceptions will abort execution.

class mozharness.base.script.**ScriptMixin**
Bases: *mozharness.base.script.PlatformMixin*

This mixin contains simple filesystem commands and the like.

It also contains some very special but very complex methods that, together with logging and config, provide the base for all scripts in this harness.

WARNING !!! This class depends entirely on *LogMixin* methods in such a way that it will only works if a class inherits from both *ScriptMixin* and *LogMixin* simultaneously.

Depends on self.config of some sort.

Attributes: env (dict): a mapping object representing the string environment. script_obj (ScriptMixin): reference to a ScriptMixin instance.

chdir(*dir_name*)

chmod(*path*, *mode*)
change *path* mode to *mode*.

Args: path (str): path whose mode will be modified. mode (hex): one of the values defined at *stat*

https://docs.python.org/2/library/os.html#os.chmod

copyfile(*src*, *dest*, *log_level='info'*, *error_level='error'*, *copystat=False*, *compress=False*)
copy or compress *src* into *dest*.

Args: src (str): filepath to copy. dest (str): filepath where to move the content to. log_level (str, optional): log level to use for normal operation. Defaults to

INFO

error_level (str, optional): log level to use on error. Defaults to *ERROR* copystat (bool, optional): whether or not to copy the files metadata.

Defaults to *False*.

compress (bool, optional): whether or not to compress the destination file. Defaults to *False*.

Returns: int: -1 on error None: on success

copytree(*src*, *dest*, *overwrite='no_overwrite'*, *log_level='info'*, *error_level='error'*)
An implementation of *shutil.copytree* that allows for *dest* to exist and implements different overwrite levels: - 'no_overwrite' will keep all(any) existing files in destination tree - 'overwrite_if_exists' will only overwrite destination paths that have

the same path names relative to the root of the src and destination tree

•'clobber' will replace the whole destination tree(clobber) if it exists

Args: src (str): directory path to move. dest (str): directory path where to move the content to. overwrite (str): string specifying the overwrite level. log_level (str, optional): log level to use for normal operation. Defaults to

INFO

error_level (str, optional): log level to use on error. Defaults to *ERROR*

Returns: int: -1 on error None: on success

download_file(*url*, *file_name=None*, *parent_dir=None*, *create_parent_dir=True*, *error_level='error'*, *exit_code=3*, *retry_config=None*)
Python wget. Download the filename at *url* into *file_name* and put it on *parent_dir*. On error log with the specified *error_level*, on fatal exit with *exit_code*. Execute all the above based on *retry_config* parameter.

Args: url (str): URL path where the file to be downloaded is located. file_name (str, optional): file_name where the file will be written to.

Defaults to urls' filename.

parent_dir (str, optional): directory where the downloaded file will be written to. Defaults to current working directory

create_parent_dir (bool, optional): create the parent directory if it doesn't exist. Defaults to *True*

error_level (str, optional): log level to use in case an error occurs. Defaults to *ERROR*

retry_config (dict, optional): key-value pairs to be passed to *self.retry*. Defaults to *None*

Returns: str: filename where the downloaded file was written to. unknown: on failure, *failure_status* is returned.

env = None

get_filename_from_url(*url*)
parse a filename base on an url.

Args: url (str): url to parse for the filename

Returns:

str: filename parsed from the url, or *netloc* network location part of the url.

get_output_from_command(*command*, *cwd=None*, *halt_on_failure=False*, *env=None*, *silent=False*, *log_level='info'*, *tmpfile_base_path='tmpfile'*, *return_type='output'*, *save_tmpfiles=False*, *throw_exception=False*, *fatal_exit_code=2*, *ignore_errors=False*, *success_codes=None*)
Similar to run_command, but where run_command is an os.system(command) analog, get_output_from_command is a *command* analog.

Less error checking by design, though if we figure out how to do it without borking the output, great.

TODO: binary mode? silent is kinda like that. TODO: since p.wait() can take a long time, optionally log something every N seconds? TODO: optionally only keep the first or last (N) line(s) of output? TODO: optionally only return the tmp_stdout_filename?

ignore_errors=True is for the case where a command might produce standard error output, but you don't particularly care; setting to True will cause standard error to be logged at DEBUG rather than ERROR

Args:

command (str | list): command or list of commands to execute and log.

cwd (str, optional): directory path from where to execute the command. Defaults to *None*.

halt_on_failure (bool, optional): whether or not to redefine the log level as *FATAL* on error. Defaults to False.

env (dict, optional): key-value of environment values to use to run the command. Defaults to None.

silent (bool, optional): whether or not to output the stdout of executing the command. Defaults to False.

log_level (str, optional): log level name to use on normal execution. Defaults to *INFO*.

tmpfile_base_path (str, optional): base path of the file to which the output will be writen to. Defaults to 'tmpfile'.

return_type (str, optional): if equal to 'output' then the complete output of the executed command is returned, otherwise the written filenames are returned. Defaults to 'output'.

save_tmpfiles (bool, optional): whether or not to save the temporary files created from the command output. Defaults to False.

throw_exception (bool, optional): whether or not to raise an exception if the return value of the command is not zero. Defaults to False.

fatal_exit_code (int, optional): call self.fatal if the return value of the command match this value.

ignore_errors (bool, optional): whether or not to change the log level to *ERROR* for the output of stderr. Defaults to False.

success_codes (int, optional): numeric value to compare against the command return value.

Returns: None: if the cwd is not a directory. None: on IOError. tuple: stdout and stderr filenames. str: stdout output.

is_exe (*fpath*)
Determine if fpath is a file and if it is executable.

mkdir_p (*path*, *error_level='error'*)
Create a directory if it doesn't exists. This method also logs the creation, error or current existence of the directory to be created.

Args: path (str): path of the directory to be created. error_level (str): log level name to be used in case of error.

Returns: None: for sucess. int: -1 on error

move (*src*, *dest*, *log_level='info'*, *error_level='error'*, *exit_code=-1*)
recursively move a file or directory (src) to another location (dest).

Args: src (str): file or directory path to move. dest (str): file or directory path where to move the content to. log_level (str): log level to use for normal operation. Defaults to

INFO

error_level (str): log level to use on error. Defaults to *ERROR*

Returns: int: 0 on success. -1 on error.

opened(**args*, ***kwds*)
Create a context manager to use on a with statement.

Args: file_path (str): filepath of the file to open. verbose (bool, optional): useless parameter, not used here.

Defaults to True.

open_mode (str, optional): open mode to use for openning the file. Defaults to *r*

error_level (str, optional): log level name to use on error. Defaults to *ERROR*

Yields:

tuple: (file object, error) pair. In case of error *None* is yielded as file object, together with the corresponding error. If there is no error, *None* is returned as the error.

query_env(*partial_env=None*, *replace_dict=None*, *purge_env=()*, *set_self_env=None*, *log_level='debug'*, *avoid_host_env=False*)
Environment query/generation method. The default, self.query_env(), will look for self.config['env'] and replace any special strings in there (%(PATH)s). It will then store it as self.env for speeding things up later.

If you specify partial_env, partial_env will be used instead of self.config['env'], and we don't save self.env as it's a one-off.

Args:

partial_env (dict, optional): key-value pairs of the name and value of different environment variables. Defaults to an empty dictionary.

replace_dict (dict, optional): key-value pairs to replace the old environment variables.

purge_env (list): environment names to delete from the final environment dictionary.

set_self_env (boolean, optional): whether or not the environment variables dictionary should be copied to *self*. Defaults to True.

log_level (str, optional): log level name to use on normal operation. Defaults to *DEBUG*.

avoid_host_env (boolean, optional): if set to True, we will not use any environment variables set on the host except PATH. Defaults to False.

Returns: dict: environment variables names with their values.

query_exe(*exe_name*, *exe_dict='exes'*, *default=None*, *return_type=None*, *error_level='fatal'*)
One way to work around PATH rewrites.

By default, return exe_name, and we'll fall through to searching os.environ["PATH"]. However, if self.config[exe_dict][exe_name] exists, return that. This lets us override exe paths via config file.

If we need runtime setting, we can build in self.exes support later.

Args: exe_name (str): name of the executable to search for. exe_dict(str, optional): name of the dictionary of executables

present in *self.config*. Defaults to *exes*.

default (str, optional): default name of the executable to search for. Defaults to *exe_name*.

return_type (str, optional): type to which the original return value will be turn into. Only 'list', 'string' and *None* are supported. Defaults to *None*.

error_level (str, optional): log level name to use on error.

Returns: list: in case return_type is 'list' str: in case return_type is 'string' None: in case return_type is *None* Any: if the found executable is not of type list, tuple nor str.

query_msys_path (*path*)
replaces the Windows harddrive letter path style with a linux path style, e.g. C:// –> /C/ Note: method, not used in any script.

Args: path (str?): path to convert to the linux path style.

Returns: str: in case *path* is a string. The result is the path with the new notation. type(path): *path* itself is returned in case *path* is not str type.

read_from_file (*file_path*, *verbose=True*, *open_mode='r'*, *error_level='error'*)
Use *self.opened* context manager to open a file and read its content.

Args: file_path (str): filepath of the file to read. verbose (bool, optional): whether or not to log the file content.

Defaults to True.

open_mode (str, optional): open mode to use for openning the file. Defaults to *r*

error_level (str, optional): log level name to use on error. Defaults to *ERROR*

Returns: None: on error. str: file content on success.

retry (*action*, *attempts=None*, *sleeptime=60*, *max_sleeptime=300*, *retry_exceptions=(<type 'exceptions.Exception'>,)*, *good_statuses=None*, *cleanup=None*, *error_level='error'*, *error_message='%(action)s failed after %(attempts)d tries!'*, *failure_status=-1*, *log_level='info'*, *args=()*, *kwargs={}*)
generic retry command. Ported from **'util.retry'_**

Args: action (func): callable object to retry. attempts (int, optinal): maximum number of times to call actions.

Defaults to *self.config.get('global_retries', 5)*

sleeptime (int, optional): number of seconds to wait between attempts. Defaults to 60 and doubles each retry attempt, to a maximum of 'max_sleeptime'

max_sleeptime (int, optional): maximum value of sleeptime. Defaults to 5 minutes

retry_exceptions (tuple, optional): Exceptions that should be caught. If exceptions other than those listed in 'retry_exceptions' are raised from 'action', they will be raised immediately. Defaults to (Exception)

good_statuses (object, optional): return values which, if specified, will result in retrying if the return value isn't listed. Defaults to *None*.

cleanup (func, optional): If 'cleanup' is provided and callable it will be called immediately after an Exception is caught. No arguments will be passed to it. If your cleanup function requires arguments it is recommended that you wrap it in an argumentless function. Defaults to *None*.

error_level (str, optional): log level name in case of error. Defaults to *ERROR*.

error_message (str, optional): string format to use in case none of the attempts success. Defaults to '%(action)s failed after %(attempts)d tries!'

failure_status (int, optional): flag to return in case the retries were not successfull. Defaults to -1.

log_level (str, optional): log level name to use for normal activity. Defaults to *INFO*.

args (tuple, optional): positional arguments to pass onto *action*. kwargs (dict, optional): key-value arguments to pass onto *action*.

Returns: object: return value of *action*. int: failure status in case of failure retries.

rmtree(*path*, *log_level='info'*, *error_level='error'*, *exit_code=-1*)

Delete an entire directory tree and log its result. This method also logs the platform rmtree function, its retries, errors, and current existence of the directory.

Args: path (str): path to the directory tree root to remove. log_level (str, optional): log level name to for this operation. Defaults

to *INFO*.

error_level (str, optional): log level name to use in case of error. Defaults to *ERROR*.

exit_code (int, optional): useless parameter, not use here. Defaults to -1

Returns: None: for success

run_command(*command*, *cwd=None*, *error_list=None*, *halt_on_failure=False*, *success_codes=None*, *env=None*, *partial_env=None*, *return_type='status'*, *throw_exception=False*, *output_parser=None*, *output_timeout=None*, *fatal_exit_code=2*, *error_level='error'*, ***kwargs*)

Run a command, with logging and error parsing. TODO: context_lines

error_list example: [{'regex': re.compile('^Error: LOL J/K'), level=IGNORE},

{'regex': re.compile('^Error:'), level=ERROR, contextLines='5:5'}, {'substr': 'THE WORLD IS ENDING', level=FATAL, contextLines='20:'}

] (context_lines isn't written yet)

Args:

command (str | list | tuple): command or sequence of commands to execute and log.

cwd (str, optional): directory path from where to execute the command. Defaults to *None*.

error_list (list, optional): list of errors to pass to *mozharness.base.log.OutputParser*. Defaults to *None*.

halt_on_failure (bool, optional): whether or not to redefine the log level as *FATAL* on errors. Defaults to False.

success_codes (int, optional): numeric value to compare against the command return value.

env (dict, optional): key-value of environment values to use to run the command. Defaults to None.

partial_env (dict, optional): key-value of environment values to replace from the current environment values. Defaults to None.

return_type (str, optional): if equal to 'num_errors' then the amount of errors matched by *error_list* is returned. Defaults to 'status'.

throw_exception (bool, optional): whether or not to raise an exception if the return value of the command doesn't match any of the *success_codes*. Defaults to False.

output_parser (OutputParser, optional): lets you provide an instance of your own OutputParser subclass. Defaults to *OutputParser*.

output_timeout (int): amount of seconds to wait for output before the process is killed.

fatal_exit_code (int, optional): call *self.fatal* if the return value of the command is not on in *success_codes*. Defaults to 2.

error_level (str, optional): log level name to use on error. Defaults to *ERROR*.

**kwargs: Arbitrary keyword arguments.

Returns: int: -1 on error. Any: *command* return value is returned otherwise.

script_obj = None

unpack(*filename*, *extract_to*)
This method allows us to extract a file regardless of its extension

Args: filename (str): filename of the compressed file. extract_to (str): where to extract the compressed file.

which(*program*)
OS independent implementation of Unix's which command

Args:

program (str): name or path to the program whose executable is being searched.

Returns: None: if the executable was not found. str: filepath of the executable file.

write_to_file(*file_path*, *contents*, *verbose=True*, *open_mode='w'*, *create_parent_dir=False*, *error_level='error'*)
Write *contents* to *file_path*, according to *open_mode*.

Args: file_path (str): filepath where the content will be written to. contents (str): content to write to the filepath. verbose (bool, optional): whether or not to log *contents* value.

Defaults to *True*

open_mode (str, optional): open mode to use for openning the file. Defaults to *w*

create_parent_dir (bool, optional): whether or not to create the parent directory of *file_path*

error_level (str, optional): log level to use on error. Defaults to *ERROR*

Returns: str: *file_path* on success None: on error.

mozharness.base.script.**platform_name**()

mozharness.base.signing module

Generic signing methods.

class mozharness.base.signing.**AndroidSigningMixin**
Bases: object

Generic Android apk signing methods.

Dependent on BaseScript.

align_apk(*unaligned_apk*, *aligned_apk*, *error_level='error'*)
Zipalign apk. Returns None on success, not None on failure.

key_passphrase = None

passphrase()

postflight_passphrase()

sign_apk(*apk*, *keystore*, *storepass*, *keypass*, *key_alias*, *remove_signature=True*, *error_list=None*, *log_level='info'*, *error_level='error'*)
Signs an apk with jarsigner.

store_passphrase = **None**

unsign_apk(*apk*, ***kwargs*)

verify_passphrases()

class mozharness.base.signing.**BaseSigningMixin**
Bases: object

Generic signing helper methods.

query_filesize(*file_path*)

query_sha512sum(*file_path*)

mozharness.base.transfer module

Generic ways to upload + download files.

class mozharness.base.transfer.**TransferMixin**
Bases: object

Generic transfer methods.

Dependent on BaseScript.

load_json_from_url(*url*, *timeout=30*, *log_level='debug'*)

rsync_download_directory(*ssh_key*, *ssh_user*, *remote_host*, *remote_path*, *local_path*, *rsync_options=None*, *error_level='error'*)
rsync+ssh the content of a remote directory to local_path

Returns:

None: on success -1: if local_path is not a directory -3: rsync fails to download from the remote directory

rsync_upload_directory(*local_path*, *ssh_key*, *ssh_user*, *remote_host*, *remote_path*, *rsync_options=None*, *error_level='error'*, *create_remote_directory=True*)
Create a remote directory and upload the contents of a local directory to it via rsync+ssh.

Returns:

None: on success -1: if local_path is not a directory -2: if the remote_directory cannot be created

(it only makes sense if create_remote_directory is True)

-3: rsync fails to copy to the remote directory

Module contents

mozharness.mozilla package

Subpackages

mozharness.mozilla.building package

Submodules

mozharness.mozilla.building.buildbase module buildbase.py.

provides a base class for fx desktop builds author: Jordan Lund

class mozharness.mozilla.building.buildbase.**BuildOptionParser**
Bases: object

bits = None

branch_cfg_file = 'builds/branch_specifics.py'

build_pool_cfg_file = 'builds/build_pool_specifics.py'

build_variants = {'api-9': 'builds/releng_sub_%s_configs/%s_api_9.py', 'api-11': 'builds/releng_sub_%s_configs/%s

config_file_search_path = ['.', '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/checko

platform = None

classmethod set_bits(*option*, *opt*, *value*, *parser*)

classmethod set_build_branch(*option*, *opt*, *value*, *parser*)

classmethod set_build_pool(*option*, *opt*, *value*, *parser*)

classmethod set_build_variant(*option*, *opt*, *value*, *parser*)
sets an extra config file.

This is done by either taking an existing filepath or by taking a valid shortname coupled with known platform/bits.

classmethod set_platform(*option*, *opt*, *value*, *parser*)

class mozharness.mozilla.building.buildbase.**BuildScript**(***kwargs*)
Bases: *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.mozilla.purge.PurgeMixin*, *mozharness.mozilla.mock.MockMixin*, mozharness.mozilla.updates.balrog.BalrogMixin, *mozharness.mozilla.signing.SigningMixin*, *mozharness.base.python.VirtualenvMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.base.transfer.TransferMixin*, *mozharness.base.python.InfluxRecordingMixin*

build()
builds application.

check_test()

checkout_sources()

clone_tools()
clones the tools repo.

generate_build_props(*console_output=True*, *halt_on_failure=False*)
sets props found from mach build and, in addition, buildid, sourcestamp, appVersion, and appName.

generate_build_stats()
grab build stats following a compile.

This action handles all statistics from a build: 'count_ctors' and then posts to graph server the results. We only post to graph server for non nightly build

multi_l10n()

package_source()
generates source archives and uploads them

postflight_build(*console_output=True*)
grabs properties from post build and calls ccache -s

preflight_build()
set up machine state for a complete build.

preflight_package_source()

query_build_env(*replace_dict=None*, ***kwargs*)

query_buildid()

query_builduid()

query_check_test_env()

query_mach_build_env(*multiLocale=None*)

query_pushdate()

query_revision(*source_path=None*)
returns the revision of the build

first will look for it in buildbot_properties and then in buildbot_config. Failing that, it will actually poll the source of the repo if it exists yet.

This method is used both to figure out what revision to check out and to figure out what revision *was* checked out.

sendchange()

update()
submit balrog update steps.

upload_files()

class mozharness.mozilla.building.buildbase.**BuildingConfig**(*config=None*, *initial_config_file=None*, *config_options=None*, *all_actions=None*, *default_actions=None*, *volatile_config=None*, *option_args=None*, *require_config_file=False*, *append_env_variables_from_configs=False*, *usage='usage: %prog [options]'*)

Bases: *mozharness.base.config.BaseConfig*

get_cfgs_from_files(*all_config_files*, *options*)
Determine the configuration from the normal options and from *–branch*, *–build-pool*, and *–custom-build-*

variant-cfg. If the files for any of the latter options are also given with *–config-file* or *–opt-config-file*, they are only parsed once.

The build pool has highest precedence, followed by branch, build variant, and any normally-specified configuration files.

class mozharness.mozilla.building.buildbase.**CheckTestCompleteParser**(***kwargs*)
Bases: *mozharness.base.log.OutputParser*

evaluate_parser()

parse_single_line(*line*)

tbpl_error_list = [{'regex': <_sre.SRE_Pattern object at 0x7fd926f5fc38>, 'level': 'RETRY'}, {'regex': <_sre.SRE_

class mozharness.mozilla.building.buildbase.**MakeUploadOutputParser**(*use_package_as_marfile=False*, *package_filename=None*, ***kwargs*)
Bases: *mozharness.base.log.OutputParser*

parse_single_line(*line*)

property_conditions = [('symbolsUrl', "m.endswith('crashreporter-symbols.zip') or m.endswith('crashreporter-sym

tbpl_error_list = [{'regex': <_sre.SRE_Pattern object at 0x7fd926f5fc38>, 'level': 'RETRY'}, {'regex': <_sre.SRE_

mozharness.mozilla.building.buildbase.**generate_build_ID**()

mozharness.mozilla.building.buildbase.**generate_build_UID**()

Module contents

mozharness.mozilla.l10n package

Submodules

mozharness.mozilla.l10n.locales module Localization.

class mozharness.mozilla.l10n.locales.**GaiaLocalesMixin**
Bases: object

gaia_locale_revisions = None

pull_gaia_locale_source(*l10n_config*, *locales*, *base_dir*)

class mozharness.mozilla.l10n.locales.**LocalesMixin**(***kwargs*)
Bases: *mozharness.base.parallel.ChunkingMixin*

list_locales()
Stub action method.

parse_locales_file(*locales_file*)

pull_locale_source(*hg_l10n_base=None*, *parent_dir=None*, *vcs='hg'*)

query_abs_dirs()

query_locales()

run_compare_locales(*locale*, *halt_on_failure=False*)

mozharness.mozilla.l10n.multi_locale_build module multi_locale_build.py

This should be a mostly generic multilocale build script.

class mozharness.mozilla.l10n.multi_locale_build.**MultiLocaleBuild**(*require_config_file=True*)
Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.base.vcs.vcsbase.Mercurial*

This class targets Fennec multilocale builds. We were considering this for potential Firefox desktop multilocale. Now that we have a different approach for B2G multilocale, it's most likely misnamed.

add_locales()

additional_packaging(*package_type='en-US'*, *env=None*)

backup_objdir()

build()

clobber()

config_options = [[['–locale'], {'action': 'extend', 'dest': 'locales', 'type': 'string', 'help': 'Specify the locale(s) to rep

package(*package_type='en-US'*)

package_en_US()

package_multi()

preflight_package_multi()

pull_build_source()

restore_objdir()

upload_en_US()

upload_multi()

Module contents

mozharness.mozilla.testing package

Submodules

mozharness.mozilla.testing.device module Interact with a device via ADB or SUT.

This code is largely from https://hg.mozilla.org/build/tools/file/default/sut_tools

class mozharness.mozilla.testing.device.**ADBDeviceHandler**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.BaseDeviceHandler*

check_device()

cleanup_device(*reboot=False*)

connect_device()

disconnect_device()

install_app(*file_path*)

ping_device(*auto_connect=False*, *silent=False*)

query_device_exe(*exe_name*)

query_device_file_exists(*file_name*)

query_device_id(*auto_connect=True*)

query_device_root(*silent=False*)

query_device_time()

reboot_device()

remove_device_root(*error_level='error'*)

remove_etc_hosts(*hosts_file='/system/etc/hosts'*)

set_device_time(*device_time=None*, *error_level='error'*)

uninstall_app(*package_name*, *package_root='/data/data'*, *error_level='error'*)

wait_for_device(*interval=60*, *max_attempts=20*)

class mozharness.mozilla.testing.device.**BaseDeviceHandler**(*log_obj=None*, *config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

add_device_flag(*flag*)

check_device()

cleanup_device(*reboot=False*)

default_port = None

device_flags = []

device_id = None

device_root = None

install_app(*file_path*)

ping_device()

query_device_id()

query_device_root()

query_download_filename(*file_id=None*)

reboot_device()

wait_for_device(*interval=60*, *max_attempts=20*)

exception mozharness.mozilla.testing.device.**DeviceException**

Bases: exceptions.Exception

class mozharness.mozilla.testing.device.**DeviceMixin**

Bases: object

BaseScript mixin, designed to interface with the device.

check_device()

cleanup_device(***kwargs*)

device_handler = None

device_root = None

install_app()

query_device_handler()

reboot_device()

class mozharness.mozilla.testing.device.**SUTDeviceHandler**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.BaseDeviceHandler*

check_device()

cleanup_device(*reboot=False*)

install_app(*file_path*)

ping_device()

query_device_root(*strict=False*)

query_device_time()

query_devicemanager()

reboot_device()

remove_etc_hosts(*hosts_file='/system/etc/hosts'*)

set_device_time()

wait_for_device(*interval=60*, *max_attempts=20*)

class mozharness.mozilla.testing.device.**SUTDeviceMozdeviceMixin**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.SUTDeviceHandler*

This SUT device manager class makes calls through mozdevice (from mozbase) [1] directly rather than calling SUT tools.

[1] https://github.com/mozilla/mozbase/blob/master/mozdevice/mozdevice/devicemanagerSUT.py

dm = None

get_logcat()

query_devicemanager()

query_file(*filename*)

set_device_epoch_time(*timestamp=1440001627*)

mozharness.mozilla.testing.errors module Mozilla error lists for running tests.

Error lists are used to parse output in mozharness.base.log.OutputParser.

Each line of output is matched against each substring or regular expression in the error list. On a match, we determine the 'level' of that line, whether IGNORE, DEBUG, INFO, WARNING, ERROR, CRITICAL, or FATAL.

mozharness.mozilla.testing.mozpool module Interact with mozpool/lifeguard/bmm.

class mozharness.mozilla.testing.mozpool.**MozpoolMixin**
Bases: object

determine_mozpool_host(*device*)

mobile_imaging_format = 'http://mobile-imaging'

mozpool_handler = None

query_mozpool_handler(*device=None*, *mozpool_api_url=None*)

retrieve_android_device(*b2gbase*)

retrieve_b2g_device(*b2gbase*)

mozharness.mozilla.testing.talos module run talos tests in a virtualenv

class mozharness.mozilla.testing.talos.**Talos**(***kwargs*)
Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadMi*

install and run Talos tests: https://wiki.mozilla.org/Buildbot/Talos

clone_talos()

config_options = [[['–talos-url'], {'action': 'store', 'dest': 'talos_url', 'default': 'https://hg.mozilla.org/build/talos/arc

create_virtualenv(***kwargs*)
VirtualenvMixin.create_virtualenv() assuemes we're using self.config['virtualenv_modules']. Since we are installing talos from its source, we have to wrap that method here.

download_talos_json()

postflight_create_virtualenv()
This belongs in download_and_install() but requires the virtualenv to be set up :(

The real fix here may be a –tpmanifest option for PerfConfigurator.

preflight_run_tests()

query_abs_dirs()

query_abs_pagesets_paths()
Returns a bunch of absolute pagesets directory paths. We need this to make the dir and copy the manifest to the local dir.

query_pagesets_manifest_filename()

query_pagesets_manifest_parent_path()

query_pagesets_manifest_path()
We have to copy the tp manifest from webroot to talos root when those two directories aren't the same, until bug 795172 is fixed.

Helper method to avoid hardcodes.

query_pagesets_parent_dir_path()
We have to copy the pageset into the webroot separately.

Helper method to avoid hardcodes.

query_pagesets_url()
Certain suites require external pagesets to be downloaded and extracted.

query_sps_profile_options()

query_talos_json_config()
Return the talos json config; download and read from the talos_json_url if need be.

query_talos_json_url()
Hacky, but I haven't figured out a better way to get the talos json url before we install the build.

We can't get this information after we install the build, because we have to create the virtualenv to use mozinstall, and talos_url is specified in the talos json.

query_talos_options()

query_talos_repo()
Where do we install the talos python package from? This needs to be overrideable by the talos json.

query_talos_revision()
Which talos revision do we want to use? This needs to be overrideable by the talos json.

query_tests()
Determine if we have tests to run.

Currently talos json will take precedence over config and command line options; if that's not a good default we can switch the order.

run_tests(*args=None*, ***kw*)
run Talos tests

talos_conf_path(*conf*)
return the full path for a talos .yml configuration file

talos_options(*args=None*, ***kw*)
return options to talos

class mozharness.mozilla.testing.talos.**TalosOutputParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)
Bases: *mozharness.base.log.OutputParser*

minidump_output = None

minidump_regex = <_sre.SRE_Pattern object at 0x35b9d50>

parse_single_line(*line*)
In Talos land, every line that starts with RETURN: needs to be printed with a TinderboxPrint:

worst_tbpl_status = 'SUCCESS'

mozharness.mozilla.testing.testbase module

class mozharness.mozilla.testing.testbase.**TestingMixin**(**args*, ***kwargs*)
Bases: *mozharness.base.python.VirtualenvMixin*, *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.base.python.ResourceMonitoringMixin*, *mozharness.mozilla.tooltool.TooltoolMixi*
mozharness.mozilla.testing.try_tools.TryToolsMixin

The steps to identify + download the proper bits for [browser] unit tests and Talos.

binary_path = None

default_tools_repo = 'https://hg.mozilla.org/build/tools'

download_and_extract(*target_unzip_dirs=None*, *suite_categories=None*)
download and extract test zip / download installer

download_file(**args*, ***kwargs*)
This function helps not to use download of proxied files since it does not support authenticated downloads. This could be re-factored and fixed in bug 1087664.

download_proxied_file(*url*, *file_name=None*, *parent_dir=None*, *create_parent_dir=True*, *error_level='fatal'*, *exit_code=3*)

get_test_output_parser(*suite_category*, *strict=False*, *fallback_parser_class=<class 'mozharness.mozilla.testing.unittest.DesktopUnittestOutputParser'>*, ***kwargs*)
Derive and return an appropriate output parser, either the structured output parser or a fallback based on the type of logging in use as determined by configuration.

install()

install_app(*app=None*, *target_dir=None*, *installer_path=None*)
Dependent on mozinstall

installer_path = None

installer_url = None

jsshell_url = None

minidump_stackwalk_path = None

postflight_read_buildbot_config()
Determine which files to download from the buildprops.json file created via the buildbot ScriptFactory.

postflight_run_tests()
preflight commands for all tests

preflight_download_and_extract()

preflight_install()

preflight_run_tests()
preflight commands for all tests

proxxy = None

query_build_dir_url(*file_name*)
Resolve a file name to a potential url in the build upload directory where that file can be found.

query_minidump_filename()

query_minidump_stackwalk()

query_minidump_tooltool_manifest()

query_symbols_url()

query_value(*key*)
This function allows us to check for a value in the self.tree_config first and then on self.config

structured_output(*suite_category*)
Defines whether structured logging is in use in this configuration. This may need to be replaced with data from a different config at the resolution of bug 1070041 and related bugs.

symbols_path = None

symbols_url = None

test_packages_url = None

test_url = None

test_zip_path = None

tree_config = {}

mozharness.mozilla.testing.unittest module

class mozharness.mozilla.testing.unittest.**DesktopUnittestOutputParser**(*suite_category*, ***kwargs*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can parse the number of passed/failed/todo tests from the output.

append_tinderboxprint_line(*suite_name*)

evaluate_parser(*return_code*, *success_codes=None*)

parse_single_line(*line*)

class mozharness.mozilla.testing.unittest.**EmulatorMixin**
Bases: object

Currently dependent on both TooltoolMixin and TestingMixin)

install_emulator()

install_emulator_from_tooltool(*manifest_path*, *do_unzip=True*)

class mozharness.mozilla.testing.unittest.**TestSummaryOutputParserHelper**(*regex=<_sre.SRE_Pattern object>*, ***kwargs*)
Bases: *mozharness.base.log.OutputParser*

evaluate_parser()

parse_single_line(*line*)

print_summary(*suite_name*)

mozharness.mozilla.testing.unittest.**tbox_print_summary**(*pass_count*, *fail_count*, *known_fail_count=None*, *crashed=False*, *leaked=False*)

Module contents

Submodules

mozharness.mozilla.blob_upload module

class mozharness.mozilla.blob_upload.**BlobUploadMixin**(**args*, ***kwargs*)
Bases: *mozharness.base.python.VirtualenvMixin*

Provides mechanism to automatically upload files written in MOZ_UPLOAD_DIR to the blobber upload server at the end of the running script.

This is dependent on ScriptMixin and BuildbotMixin. The testing script inheriting this class is to specify as cmdline options the <blob-upload-branch> and <blob-upload-server>

upload_blobber_files()

mozharness.mozilla.buildbot module

Code to tie into buildbot. Ideally this will go away if and when we retire buildbot.

class mozharness.mozilla.buildbot.**BuildbotMixin**
Bases: object

buildbot_config = None

buildbot_properties = {}

buildbot_status(*tbpl_status*, *level=None*, *set_return_code=True*)

dump_buildbot_properties(*prop_list=None*, *file_name='properties'*, *error_level='error'*)

invoke_sendchange(*downloadables=None*, *branch=None*, *username='sendchange-unittest'*, *sendchange_props=None*)
Generic sendchange, currently b2g- and unittest-specific.

query_buildbot_property(*prop_name*)

query_is_nightly()
returns whether or not the script should run as a nightly build.

First will check for 'nightly_build' in self.config and if that is not True, we will also allow buildbot_config to determine for us. Failing all of that, we default to False. Note, dependancy on buildbot_config is being deprecated. Putting everything in self.config is the preference.

read_buildbot_config()

set_buildbot_property(*prop_name*, *prop_value*, *write_to_file=False*)

tryserver_email()

worst_buildbot_status = 'SUCCESS'

mozharness.mozilla.gaia module

Module for performing gaia-specific tasks

class mozharness.mozilla.gaia.**GaiaMixin**
Bases: object

clone_gaia(*dest*, *repo*, *use_gaia_json=False*)
Clones an hg mirror of gaia.

repo: a dict containing 'repo_path', 'revision', and optionally 'branch' parameters

use_gaia_json: if True, the repo parameter is used to retrieve a gaia.json file from a gecko repo, which in turn is used to clone gaia; if False, repo represents a gaia repo to clone.

extract_xre(*xre_url*, *xre_path=None*, *parent_dir=None*)

make_gaia(*gaia_dir*, *xre_dir*, *debug=False*, *noftu=True*, *xre_url=None*, *build_config_path=None*)

make_node_modules()

node_setup()
Set up environment for node-based Gaia tests.

npm_error_list = [{'substr': 'command not found', 'level': 'error'}, {'substr': 'npm ERR! Error:', 'level': 'error'}]

preflight_pull()

pull(***kwargs*)
Two ways of using this function: - The user specifies –gaia-repo or in a config file - The buildbot propeties exist and we query the gaia json url

for the current gecko tree

mozharness.mozilla.mapper module

Support for hg/git mapper

class mozharness.mozilla.mapper.**MapperMixin**

query_mapper(*mapper_url*, *project*, *vcs*, *rev*, *require_answer=True*, *attempts=30*, *sleeptime=30*, *project_name=None*)
Returns the mapped revision for the target vcs via a mapper service

Args: mapper_url (str): base url to use for the mapper service project (str): The name of the mapper project to use for lookups vcs (str): Which vcs you want the revision for. e.g. "git" to get

the git revision given an hg revision

rev (str): The original revision you want the mapping for. require_answer (bool): Whether you require a valid answer or not.

If None is acceptable (meaning mapper doesn't know about the revision you're asking about), then set this to False. If True, then will return the revision, or cause a fatal error.

attempts (int): How many times to try to do the lookup sleeptime (int): How long to sleep between attempts project_name (str): Used for logging only to give a more

descriptive name to the project, otherwise just uses the project parameter

Returns: A revision string, or None

query_mapper_git_revision(*url*, *project*, *rev*, ***kwargs*)
Returns the git revision for the given hg revision *rev* See query_mapper docs for supported parameters and docstrings

query_mapper_hg_revision(*url*, *project*, *rev*, ***kwargs*)
Returns the hg revision for the given git revision *rev* See query_mapper docs for supported parameters and docstrings

mozharness.mozilla.mock module

Code to integrate with mock

class mozharness.mozilla.mock.**MockMixin**
Bases: object

Provides methods to setup and interact with mock environments. https://wiki.mozilla.org/ReleaseEngineering/Applications/Mock

This is dependent on ScriptMixin

copy_mock_files(*mock_target*, *files*)
Copy files into the mock environment *mock_target*. *files* should be an iterable of 2-tuples: (src, dst)

default_mock_target = None

delete_mock_files(*mock_target*, *files*)
Delete files from the mock environment *mock_target*. *files* should be an iterable of 2-tuples: (src, dst). Only the dst component is deleted.

disable_mock()
Restore self.run_command and self.get_output_from_command to their original versions. This is the opposite of self.enable_mock()

done_mock_setup = False

enable_mock()
Wrap self.run_command and self.get_output_from_command to run inside the mock environment given by self.config['mock_target']

get_mock_output_from_command(*mock_target*, *command*, *cwd=None*, *env=None*, ***kwargs*)
Same as ScriptMixin.get_output_from_command, except runs command inside mock environment *mock_target*.

get_mock_target()

get_output_from_command_m(**args*, ***kwargs*)
Executes self.get_mock_output_from_command if we have a mock target set, otherwise executes self.get_output_from_command.

init_mock(*mock_target*)
Initialize mock environment defined by *mock_target*

install_mock_packages(*mock_target*, *packages*)
Install *packages* into mock environment *mock_target*

mock_enabled = False

reset_mock(*mock_target=None*)
rm mock lock and reset

run_command_m(**args*, ***kwargs*)
Executes self.run_mock_command if we have a mock target set, otherwise executes self.run_command.

run_mock_command(*mock_target*, *command*, *cwd=None*, *env=None*, ***kwargs*)
Same as ScriptMixin.run_command, except runs command inside mock environment *mock_target*.

setup_mock(*mock_target=None*, *mock_packages=None*, *mock_files=None*)
Initializes and installs packages, copies files into mock environment given by configuration in self.config. The mock environment is given by self.config['mock_target'], the list of packges to install given by self.config['mock_packages'], and the list of files to copy in is self.config['mock_files'].

mozharness.mozilla.mozbase module

class mozharness.mozilla.mozbase.**MozbaseMixin**(**args*, ***kwargs*)
Bases: object

Automatically set virtualenv requirements to use mozbase from test package.

mozharness.mozilla.purge module

Purge/clobber support

class mozharness.mozilla.purge.**PurgeMixin**
Bases: object

clobber(*always_clobber_dirs=None*)
Mozilla clobberer-type clobber.

clobber_tool = '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/checkouts/latest/external_

clobberer()

default_maxage = 14

default_periodic_clobber = 168

default_skips = ['info', 'rel-*', 'tb-rel-*']

purge_builds(*basedirs=None*, *min_size=None*, *skip=None*, *max_age=None*)

purge_tool = '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/checkouts/latest/external_too

mozharness.mozilla.release module

release.py

class mozharness.mozilla.release.**ReleaseMixin**

query_release_config()

release_config = {}

mozharness.mozilla.repo_manifest module

Module for handling repo style XML manifests

mozharness.mozilla.repo_manifest.**add_project**(*manifest*, *name*, *path*, *remote=None*, *revision=None*)
Adds a project to the manifest in place

mozharness.mozilla.repo_manifest.**cleanup**(*manifest*, *depth=0*)
Remove any empty text nodes

mozharness.mozilla.repo_manifest.**get_default**(*manifest*)

mozharness.mozilla.repo_manifest.**get_project**(*manifest*, *name=None*, *path=None*)
Gets a project node from the manifest. One of name or path must be set. If path is specified, then the project with the given path is returned, otherwise the project with the given name is returned.

mozharness.mozilla.repo_manifest.**get_project_remote_url**(*manifest*, *project*)
Gets the remote URL for the given project node. Will return the default remote if the project doesn't explicitly specify one.

mozharness.mozilla.repo_manifest.**get_project_revision**(*manifest*, *project*)
Gets the revision for the given project node. Will return the default revision if the project doesn't explicitly specify one.

mozharness.mozilla.repo_manifest.**get_remote**(*manifest*, *name*)

mozharness.mozilla.repo_manifest.**is_commitid**(*revision*)
Returns True if revision looks like a commit id i.e. 40 character string made up of 0-9a-f

mozharness.mozilla.repo_manifest.**load_manifest**(*filename*)
Loads manifest from *filename* and returns a single flattened manifest Processes any <include name="..." /> nodes recursively Removes projects referenced by <remove-project name="..." /> nodes Abort on unsupported manifest tags Returns the root node of the resulting DOM

mozharness.mozilla.repo_manifest.**map_remote**(*r*, *mappings*)
Helper function for mapping git remotes

mozharness.mozilla.repo_manifest.**remove_group**(*manifest*, *group*)
Removes all projects with groups='group'

mozharness.mozilla.repo_manifest.**remove_project**(*manifest*, *name=None*, *path=None*)
Removes a project from manifest. One of name or path must be set. If path is specified, then the project with the given path is removed, otherwise the project with the given name is removed.

mozharness.mozilla.repo_manifest.**rewrite_remotes**(*manifest*, *mapping_func*, *force_all=True*)
Rewrite manifest remotes in place Returns the same manifest, with the remotes transformed by mapping_func mapping_func should return a modified remote node, or None if no changes are required If force_all is True, then it is an error for mapping_func to return None; a ValueError is raised in this case

mozharness.mozilla.signing module

Mozilla-specific signing methods.

class mozharness.mozilla.signing.**MobileSigningMixin**
Bases: *mozharness.base.signing.AndroidSigningMixin*, *mozharness.mozilla.signing.SigningMixi*

verify_android_signature(*apk*, *script=None*, *key_alias='nightly'*, *tools_dir='tools/'*, *env=None*)
Runs mjessome's android signature verification script. This currently doesn't check to see if the apk exists; you may want to do that before calling the method.

class mozharness.mozilla.signing.**SigningMixin**
Bases: *mozharness.base.signing.BaseSigningMixin*

Generic signing helper methods.

query_moz_sign_cmd(*formats='gpg'*)

mozharness.mozilla.tooltool module

module for tooltool operations

class mozharness.mozilla.tooltool.**TooltoolMixin**
Bases: object

Mixin class for handling tooltool manifests. To use a tooltool server other than the Mozilla server, override config['tooltool_servers']. To specify a different authentication file than that used in releng automation,override config['tooltool_authentication_file']; set it to None to not pass any authentication information (OK for public files)

create_tooltool_manifest(*contents*, *path=None*)
Currently just creates a manifest, given the contents. We may want a template and individual values in the future?

tooltool_fetch(*manifest*, *bootstrap_cmd=None*, *output_dir=None*, *privileged=False*, *cache=None*)
docstring for tooltool_fetch

Module contents

1.1.2 Module contents

1.2 mozharness.base package

1.2.1 Subpackages

mozharness.base.vcs package

Submodules

mozharness.base.vcs.gittool module

class mozharness.base.vcs.gittool.**GittoolParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can find the "Got revision" string from gittool.py output

got_revision = None

got_revision_exp = <_sre.SRE_Pattern object>

parse_single_line(*line*)

class mozharness.base.vcs.gittool.**GittoolVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

mozharness.base.vcs.hgtool module

class mozharness.base.vcs.hgtool.**HgtoolParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can find the "Got revision" string from hgtool.py output

got_revision = None

got_revision_exp = <_sre.SRE_Pattern object>

parse_single_line(*line*)

class mozharness.base.vcs.hgtool.**HgtoolVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

mozharness.base.vcs.mercurial module

Mercurial VCS support.

Largely copied/ported from https://hg.mozilla.org/build/tools/file/cf265ea8fb5e/lib/python/util/hg.py .

class mozharness.base.vcs.mercurial.**MercurialVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*, object

apply_and_push(*localrepo*, *remote*, *changer*, *max_attempts=10*, *ssh_username=None*, *ssh_key=None*)

This function calls 'changer' to make changes to the repo, and tries its hardest to get them to the origin repo. 'changer' must be a callable object that receives two arguments: the directory of the local repository, and the attempt number. This function will push ALL changesets missing from remote.

cleanOutgoingRevs(*reponame*, *remote*, *username*, *sshKey*)

clone(*repo*, *dest*, *branch=None*, *revision=None*, *update_dest=True*)

Clones hg repo and places it at *dest*, replacing whatever else is there. The working copy will be empty.

If *revision* is set, only the specified revision and its ancestors will be cloned. If revision is set, branch is ignored.

If *update_dest* is set, then *dest* will be updated to *revision* if set, otherwise to *branch*, otherwise to the head of default.

common_args(*revision=None*, *branch=None*, *ssh_username=None*, *ssh_key=None*)

Fill in common hg arguments, encapsulating logic checks that depend on mercurial versions and provided arguments

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

get_branch_from_path(*path*)

get_branches_from_path(*path*)

get_repo_name(*repo*)

get_repo_path(*repo*)

get_revision_from_path(*path*)

Returns which revision directory *path* currently has checked out.

hg_ver()

Returns the current version of hg, as a tuple of (major, minor, build)

out(*src*, *remote*, ***kwargs*)

Check for outgoing changesets present in a repo

pull(*repo*, *dest*, *update_dest=True*, ***kwargs*)

Pulls changes from hg repo and places it in *dest*.

If *revision* is set, only the specified revision and its ancestors will be pulled.

If *update_dest* is set, then *dest* will be updated to *revision* if set, otherwise to *branch*, otherwise to the head of default.

push(*src*, *remote*, *push_new_branches=True*, ***kwargs*)

query_can_share()

share(*source*, *dest*, *branch=None*, *revision=None*)
Creates a new working directory in "dest" that shares history with "source" using Mercurial's share extension

update(*dest*, *branch=None*, *revision=None*)
Updates working copy *dest* to *branch* or *revision*. If revision is set, branch will be ignored. If neither is set then the working copy will be updated to the latest revision on the current branch. Local changes will be discarded.

mozharness.base.vcs.mercurial.**make_hg_url**(*hg_host*, *repo_path*, *protocol='http'*, *revision=None*, *filename=None*)
Helper function.

Construct a valid hg url from a base hg url (hg.mozilla.org), repo_path, revision and possible filename

mozharness.base.vcs.vcsbase module

Generic VCS support.

class mozharness.base.vcs.vcsbase.**MercurialScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSScript*

default_vcs = 'hg'

class mozharness.base.vcs.vcsbase.**VCSMixin**
Bases: object

Basic VCS methods that are vcs-agnostic. The vcs_class handles all the vcs-specific tasks.

query_dest(*kwargs*)

vcs_checkout(*vcs=None*, *error_level='fatal'*, ***kwargs*)
Check out a single repo.

vcs_checkout_repos(*repo_list*, *parent_dir=None*, *tag_override=None*, ***kwargs*)
Check out a list of repos.

class mozharness.base.vcs.vcsbase.**VCSScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSMixin*, *mozharness.base.script.BaseScript*

pull(*repos=None*, *parent_dir=None*)

mozharness.base.vcs.vcssync module

Generic VCS support.

class mozharness.base.vcs.vcssync.**VCSSyncScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSScript*

notify(*message=None*, *fatal=False*)
Email people in the notify_config (depending on status and failure_only)

start_time = 1440001632.71014

Module contents

1.2.2 Submodules

1.2.3 mozharness.base.config module

Generic config parsing and dumping, the way I remember it from scripts gone by.

The config should be built from script-level defaults, overlaid by config-file defaults, overlaid by command line options.

> **(For buildbot-analogues that would be factory-level defaults,** builder-level defaults, and build request/scheduler settings.)

The config should then be locked (set to read-only, to prevent runtime alterations). Afterwards we should dump the config to a file that is uploaded with the build, and can be used to debug or replicate the build at a later time.

TODO:

- check_required_settings or something – run at init, assert that these settings are set.

class mozharness.base.config.**BaseConfig**(*config=None*, *initial_config_file=None*, *config_options=None*, *all_actions=None*, *default_actions=None*, *volatile_config=None*, *option_args=None*, *require_config_file=False*, *append_env_variables_from_configs=False*, *usage='usage: %prog [options]'*)

Bases: object

Basic config setting/getting.

get_actions()

get_cfgs_from_files(*all_config_files*, *options*)

Returns the configuration derived from the list of configuration files. The result is represented as a list of *(filename, config_dict)* tuples; they will be combined with keys in later dictionaries taking precedence over earlier.

all_config_files is all files specified with *–config-file* and *–opt-config-file*; *options* is the argparse options object giving access to any other command-line options.

This function is also responsible for downloading any configuration files specified by URL. It uses parse_config_file in this module to parse individual files.

This method can be overridden in a subclass to add extra logic to the way that self.config is made up. See *mozharness.mozilla.building.buildbase.BuildingConfig* for an example.

get_read_only_config()

list_actions()

parse_args(*args=None*)

Parse command line arguments in a generic way. Return the parser object after adding the basic options, so child objects can manipulate it.

set_config(*config*, *overwrite=False*)

This is probably doable some other way.

verify_actions(*action_list*, *quiet=False*)

verify_actions_order(*action_list*)

class mozharness.base.config.**ExtendOption**(**opts*, ***attrs*)
Bases: optparse.Option

from http://docs.python.org/library/optparse.html?highlight=optparse#adding-new-actions

ACTIONS = ('store', 'store_const', 'store_true', 'store_false', 'append', 'append_const', 'count', 'callback', 'help', 'versio

ALWAYS_TYPED_ACTIONS = ('store', 'append', 'extend')

STORE_ACTIONS = ('store', 'store_const', 'store_true', 'store_false', 'append', 'append_const', 'count', 'extend')

TYPED_ACTIONS = ('store', 'append', 'callback', 'extend')

take_action(*action*, *dest*, *opt*, *value*, *values*, *parser*)

class mozharness.base.config.**ExtendedOptionParser**(***kwargs*)
Bases: optparse.OptionParser

OptionParser, but with ExtendOption as the option_class.

class mozharness.base.config.**LockedTuple**
Bases: tuple

class mozharness.base.config.**ReadOnlyDict**(*dictionary*)
Bases: dict

clear(**args*)

lock()

pop(**args*)

popitem(**args*)

setdefault(**args*)

update(**args*)

mozharness.base.config.**download_config_file**(*url*, *file_name*)

mozharness.base.config.**make_immutable**(*item*)

mozharness.base.config.**parse_config_file**(*file_name*, *quiet=False*, *search_path=None*, *config_dict_name='config'*)
Read a config file and return a dictionary.

1.2.4 mozharness.base.errors module

Generic error lists.

Error lists are used to parse output in mozharness.base.log.OutputParser.

Each line of output is matched against each substring or regular expression in the error list. On a match, we determine the 'level' of that line, whether IGNORE, DEBUG, INFO, WARNING, ERROR, CRITICAL, or FATAL.

TODO: Context lines (requires work on the OutputParser side)

TODO: We could also create classes that generate these, but with the appropriate level (please don't die on any errors; please die on any warning; etc.) or platform or language or whatever.

exception mozharness.base.errors.**VCSException**
Bases: exceptions.Exception

1.2.5 mozharness.base.gaia_test module

1.2.6 mozharness.base.log module

Generic logging classes and functionalities for single and multi file logging. Capturing console output and providing general logging functionalities.

Attributes:

FATAL_LEVEL (int): constant logging level value set based on the logging.CRITICAL value

DEBUG (str): mozharness *debug* log name INFO (str): mozharness *info* log name WARNING (str): mozharness *warning* log name CRITICAL (str): mozharness *critical* log name FATAL (str): mozharness *fatal* log name IGNORE (str): mozharness *ignore* log name LOG_LEVELS (dict): mapping of the mozharness log level names to logging values ROOT_LOGGER (logging.Logger): instance of a logging.Logger class

TODO: - network logging support. - log rotation config

class mozharness.base.log.**BaseLogger**(*log_level='info'*, *log_format='%(message)s'*, *log_date_format='%H:%M:%S'*, *log_name='test'*, *log_to_console=True*, *log_dir='.'*, *log_to_raw=False*, *logger_name=''*, *append_to_log=False*)

Bases: object

Base class in charge of logging handling logic such as creating logging files, dirs, attaching to the console output and managing its output.

Attributes: LEVELS (dict): flat copy of the *LOG_LEVELS* attribute of the *log* module.

TODO: status? There may be a status object or status capability in either logging or config that allows you to count the number of error,critical,fatal messages for us to count up at the end (aiming for 0).

LEVELS = {'info': 20, 'warning': 30, 'critical': 50, 'error': 40, 'debug': 10, 'fatal': 60}

add_console_handler(*log_level=None*, *log_format=None*, *date_format=None*)

create a *logging.StreamHandler* using *sys.stderr* for logging the console output and add it to the *all_handlers* member variable

Args:

log_level (str, optional): useless argument. Not used here. Defaults to None.

log_format (str, optional): format used for the Formatter attached to the StreamHandler. Defaults to None.

date_format (str, optional): format used for the Formatter attached to the StreamHandler. Defaults to None.

add_file_handler(*log_path*, *log_level=None*, *log_format=None*, *date_format=None*)

create a *logging.FileHandler* base on the path, log and date format and add it to the *all_handlers* member variable.

Args: log_path (str): filepath to use for the *FileHandler*. log_level (str, optional): useless argument. Not used here.

Defaults to None.

log_format (str, optional): log format to use for the Formatter constructor. Defaults to the current instance log format.

date_format (str, optional): date format to use for the Formatter constructor. Defaults to the current instance date format.

create_log_dir()
create a logging directory if it doesn't exits. If there is a file with same name as the future logging directory it will be deleted.

get_log_formatter(*log_format=None*, *date_format=None*)
create a *logging.Formatter* base on the log and date format.

Args:

log_format (str, optional): log format to use for the Formatter constructor. Defaults to the current instance log format.

date_format (str, optional): date format to use for the Formatter constructor. Defaults to the current instance date format.

Returns: logging.Formatter: instance created base on the passed arguments

get_logger_level(*level=None*)

translate the level name passed to it and return its numeric value according to *LEVELS* values.

Args:

level (str, optional): level name to be translated. Defaults to the current instance *log_level*.

Returns:

int: numeric value of the log level name passed to it or 0 (NOTSET) if the name doesn't exists

init_message(*name=None*)
log an init message stating the name passed to it, the current date and time and, the current working directory.

Args:

name (str, optional): name to use for the init log message. Defaults to the current instance class name.

log_message(*message*, *level='info'*, *exit_code=-1*, *post_fatal_callback=None*)

Generic log method. There should be more options here – do or don't split by line, use os.linesep instead of assuming

, be able to pass in log level by name or number.

Adding the IGNORE special level for runCommand.

Args: message (str): message to log using the current *logger* level (str, optional): log level of the message. Defaults to INFO. exit_code (int, optional): exit code to use in case of a FATAL level is used.

Defaults to -1.

post_fatal_callback (function, optional): function to callback in case of of a fatal log level. Defaults None.

new_logger()
Create a new logger based on the ROOT_LOGGER instance. By default there are no handlers. The new logger becomes a member variable of the current instance as *self.logger*.

class mozharness.base.log.**LogMixin**
Bases: object

This is a mixin for any object to access similar logging functionality

The logging functionality described here is specially useful for those objects with self.config and self.log_obj member variables

critical(*message*)
calls the log method with CRITICAL as logging level

Args: message (str): message to log

debug(*message*)
calls the log method with DEBUG as logging level

Args: message (str): message to log

error(*message*)
calls the log method with ERROR as logging level

Args: message (str): message to log

exception(*message=None*, *level='error'*)
log an exception message base on the log level passed to it.

This function fetches the information of the current exception being handled and adds it to the message argument.

Args:

message (str, optional): message to be printed at the beginning of the log. Default to an empty string.

level (str, optional): log level to use for the logging. Defaults to ERROR

Returns: None

fatal(*message*, *exit_code=-1*)
calls the log method with FATAL as logging level

Args: message (str): message to log exit_code (int, optional): exit code to use for the SystemExit

exception to be raised. Default to -1.

info(*message*)
calls the log method with INFO as logging level

Args: message (str): message to log

log(*message*, *level='info'*, *exit_code=-1*)
log the message passed to it according to level, exit if level == FATAL

Args: message (str): message to be logged level (str, optional): logging level of the message. Defaults to INFO exit_code (int, optional): exit code to log before the scripts calls

SystemExit.

Returns: None

warning(*message*)
calls the log method with WARNING as logging level

Args: message (str): message to log

worst_level(*target_level*, *existing_level*, *levels=None*)
Compare target_level with existing_level according to levels values and return the worst among them.

Args:

target_level (str): minimum logging level to which the current object should be set

existing_level (str): current logging level levels (list(str), optional): list of logging levels names to compare

target_level and existing_level against. Defaults to mozharness log level list sorted from most to less critical.

Returns:

str: the logging lavel that is closest to the first levels value, i.e. levels[0]

class mozharness.base.log.**MultiFileLogger**(*logger_name='Multi'*, *log_format='%(asctime)s %(levelname)8s - %(message)s'*, *log_dir='logs'*, *log_to_raw=True*, ***kwargs*)

Bases: *mozharness.base.log.BaseLogger*

Subclass of the BaseLogger class. Create a log per log level in log_dir. Possibly also output to the terminal and a raw log (no prepending of level or date)

new_logger()

calls the BaseLogger.new_logger method and adds a file handler per logging level in the *LEVELS* class attribute.

class mozharness.base.log.**OutputParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.LogMixin*

Helper object to parse command output.

This will buffer output if needed, so we can go back and mark [(linenum - 10) : linenum+10] as errors if need be, without having to get all the output first.

linenum+10 will be easy; we can set self.num_post_context_lines to 10, and self.num_post_context_lines– as we mark each line to at least error level X.

linenum-10 will be trickier. We'll not only need to save the line itself, but also the level that we've set for that line previously, whether by matching on that line, or by a previous line's context. We should only log that line if all output has ended (self.finish() ?); otherwise store a list of dictionaries in self.context_buffer that is buffered up to self.num_pre_context_lines (set to the largest pre-context-line setting in error_list.)

add_lines(*output*)

process a string or list of strings, decode them to utf-8,strip them of any trailing whitespaces and parse them using *parse_single_line*

strings consisting only of whitespaces are ignored.

Args: output (str | list): string or list of string to parse

parse_single_line(*line*)

parse a console output line and check if it matches one in *error_list*, if so then log it according to *log_output*.

Args: line (str): command line output to parse.

class mozharness.base.log.**SimpleFileLogger**(*log_format='%(asctime)s %(levelname)8s - %(message)s'*, *logger_name='Simple'*, *log_dir='logs'*, ***kwargs*)

Bases: *mozharness.base.log.BaseLogger*

Subclass of the BaseLogger.

Create one logFile. Possibly also output to the terminal and a raw log (no prepending of level or date)

new_logger()

calls the BaseLogger.new_logger method and adds a file handler to it.

mozharness.base.log.**numeric_log_level**(*level*)
Converts a mozharness log level (string) to the corresponding logger level (number). This function makes possible to set the log level in functions that do not inherit from LogMixin

Args: level (str): log level name to convert.

Returns: int: numeric value of the log level name.

1.2.7 mozharness.base.mar module

1.2.8 mozharness.base.parallel module

Generic ways to parallelize jobs.

class mozharness.base.parallel.**ChunkingMixin**
Bases: object

Generic signing helper methods.

query_chunked_list(*possible_list*, *this_chunk*, *total_chunks*, *sort=False*)
Split a list of items into a certain number of chunks and return the subset of that will occur in this chunk.

Ported from build.l10n.getLocalesForChunk in build/tools.

1.2.9 mozharness.base.python module

Python usage, esp. virtualenv.

class mozharness.base.python.**InfluxRecordingMixin**
Bases: object

Provides InfluxDB stat recording to scripts.

This class records stats to an InfluxDB server, if enabled. Stat recording is enabled in a script by inheriting from this class, and adding an influxdb_credentials line to the influx_credentials_file (usually oauth.txt in automation). This line should look something like:

influxdb_credentials = 'http://goldiewilson-onepointtwentyone-1.c.influxdb.com:8086/db/DBNAME/series?u=DBUSERNA

Where DBNAME, DBUSERNAME, and DBPASSWORD correspond to the database name, and user/pw credentials for recording to the database. The stats from mozharness are recorded in the 'mozharness' table.

influxdb_recording_init()

influxdb_recording_post_action(*action*, *success=None*)

influxdb_recording_pre_action(*action*)

record_influx_stat(*json_data*)

record_mach_stats(*action*, *success=None*)

class mozharness.base.python.**ResourceMonitoringMixin**(**args*, ***kwargs*)
Bases: object

Provides resource monitoring capabilities to scripts.

When this class is in the inheritance chain, resource usage stats of the executing script will be recorded.

This class requires the VirtualenvMixin in order to install a package used for recording resource usage.

While we would like to record resource usage for the entirety of a script, since we require an external package, we can only record resource usage after that package is installed (as part of creating the virtualenv). That's just the way things have to be.

class mozharness.base.python.**VirtualenvMixin**(**args*, ***kwargs*)
Bases: object

BaseScript mixin, designed to create and use virtualenvs.

Config items:

- virtualenv_path points to the virtualenv location on disk.
- virtualenv_modules lists the module names.
- MODULE_url list points to the module URLs (optional)

Requires virtualenv to be in PATH. Depends on ScriptMixin

activate_virtualenv()
Import the virtualenv's packages into this Python interpreter.

create_virtualenv(*modules=()*, *requirements=()*)
Create a python virtualenv.

The virtualenv exe can be defined in c['virtualenv'] or c['exes']['virtualenv'], as a string (path) or list (path + arguments).

c['virtualenv_python_dll'] is an optional config item that works around an old windows virtualenv bug.

virtualenv_modules can be a list of module names to install, e.g.

virtualenv_modules = ['module1', 'module2']

or it can be a heterogeneous list of modules names and dicts that define a module by its name, url-or-path, and a list of its global options.

virtualenv_modules = [

{ 'name': 'module1', 'url': None, 'global_options': ['–opt', '–without-gcc']

}, {

'name': 'module2', 'url': 'http://url/to/package', 'global_options': ['–use-clang']

}, {

'name': 'module3', 'url': os.path.join('path', 'to', 'setup_py', 'dir') 'global_options': []

}, 'module4'

]

virtualenv_requirements is an optional list of pip requirements files to use when invoking pip, e.g.,

virtualenv_requirements = ['/path/to/requirements1.txt', '/path/to/requirements2.txt'

]

install_module(*module=None*, *module_url=None*, *install_method=None*, *requirements=()*, *optional=False*, *global_options=[]*, *no_deps=False*, *editable=False*)
Install module via pip.

module_url can be a url to a python package tarball, a path to a directory containing a setup.py (absolute or relative to work_dir) or None, in which case it will default to the module name.

requirements is a list of pip requirements files. If specified, these will be combined with the module_url (if any), like so:

pip install -r requirements1.txt -r requirements2.txt module_url

is_python_package_installed(*package_name*, *error_level='warning'*)
Return whether the package is installed

package_versions(*pip_freeze_output=None*, *error_level='warning'*, *log_output=False*)
reads packages from *pip freeze* output and returns a dict of {package_name: 'version'}

python_paths = {}

query_python_path(*binary='python'*)
Return the path of a binary inside the virtualenv, if c['virtualenv_path'] is set; otherwise return the binary name. Otherwise return None

query_python_site_packages_path()

query_virtualenv_path()

register_virtualenv_module(*name=None*, *url=None*, *method=None*, *requirements=None*, *optional=False*, *two_pass=False*, *editable=False*)
Register a module to be installed with the virtualenv.

This method can be called up until create_virtualenv() to register modules that should be installed in the virtualenv.

See the documentation for install_module for how the arguments are applied.

site_packages_path = None

1.2.10 mozharness.base.script module

Generic script objects.

script.py, along with config.py and log.py, represents the core of mozharness.

class mozharness.base.script.**BaseScript**(*config_options=None*, *ConfigClass=<class 'mozharness.base.config.BaseConfig'>*, *default_log_level='info'*, ***kwargs*)
Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*, object

action_message(*message*)

add_failure(*key*, *message='%(key)s failed.'*, *level='error'*, *increment_return_code=True*)

add_summary(*message*, *level='info'*)

clobber()
Delete the working directory

copy_logs_to_upload_dir()
Copies logs to the upload directory

copy_to_upload_dir(*target*, *dest=None*, *short_desc='unknown'*, *long_desc='unknown'*, *log_level='debug'*, *error_level='error'*, *max_backups=None*, *compress=False*, *upload_dir=None*)
Copy target file to upload_dir/dest.

Potentially update a manifest in the future if we go that route.

Currently only copies a single file; would be nice to allow for recursive copying; that would probably done by creating a helper _copy_file_to_upload_dir().

short_desc and long_desc are placeholders for if/when we add upload_dir manifests.

dump_config(*file_path=None*, *config=None*, *console_output=True*, *exit_on_finish=False*)
Dump self.config to localconfig.json

file_sha512sum(*file_path*)

new_log_obj(*default_log_level='info'*)

query_abs_dirs()
We want to be able to determine where all the important things are. Absolute paths lend themselves well to this, though I wouldn't be surprised if this causes some issues somewhere.

This should be overridden in any script that has additional dirs to query.

The query_* methods tend to set self.VAR variables as their runtime cache.

query_failure(*key*)

return_code

run()
Default run method. This is the "do everything" method, based on actions and all_actions.

First run self.dump_config() if it exists. Second, go through the list of all_actions. If they're in the list of self.actions, try to run self.preflight_ACTION(), self.ACTION(), and self.postflight_ACTION().

Preflight is sanity checking before doing anything time consuming or destructive.

Postflight is quick testing for success after an action.

run_action(*action*)

run_and_exit()
Runs the script and exits the current interpreter.

summarize_success_count(*success_count*, *total_count*, *message='%d of %d successful.'*, *level=None*)

summary()
Print out all the summary lines added via add_summary() throughout the script.

I'd like to revisit how to do this in a prettier fashion.

class mozharness.base.script.**PlatformMixin**
Bases: object

mozharness.base.script.**PostScriptAction**(*action=None*)
Decorator for methods that will be called at the end of each action.

This behaves similarly to PreScriptAction. It varies in that it is called after execution of the action.

The decorated method will receive the action name as a positional argument. It will then receive the following named arguments:

success - Bool indicating whether the action finished successfully.

The decorated method will always be called, even if the action threw an exception.

The return value is ignored.

mozharness.base.script.**PostScriptRun**(*func*)
Decorator for methods that will be called after script execution.

This is similar to PreScriptRun except it is called at the end of execution. The method will always be fired, even if execution fails.

mozharness.base.script.**PreScriptAction**(*action=None*)
Decorator for methods that will be called at the beginning of each action.

Each method on a BaseScript having this decorator will be called during BaseScript.run() before an individual action is executed. The method will receive the action's name as an argument.

If no values are passed to the decorator, it will be applied to every action. If a string is passed, the decorated function will only be called for the action of that name.

The return value of the method is ignored. Exceptions will abort execution.

mozharness.base.script.**PreScriptRun**(*func*)
Decorator for methods that will be called before script execution.

Each method on a BaseScript having this decorator will be called at the beginning of BaseScript.run().

The return value is ignored. Exceptions will abort execution.

class mozharness.base.script.**ScriptMixin**
Bases: *mozharness.base.script.PlatformMixin*

This mixin contains simple filesystem commands and the like.

It also contains some very special but very complex methods that, together with logging and config, provide the base for all scripts in this harness.

WARNING !!! This class depends entirely on *LogMixin* methods in such a way that it will only works if a class inherits from both *ScriptMixin* and *LogMixin* simultaneously.

Depends on self.config of some sort.

Attributes: env (dict): a mapping object representing the string environment. script_obj (ScriptMixin): reference to a ScriptMixin instance.

chdir(*dir_name*)

chmod(*path*, *mode*)
change *path* mode to *mode*.

Args: path (str): path whose mode will be modified. mode (hex): one of the values defined at *stat*

https://docs.python.org/2/library/os.html#os.chmod

copyfile(*src*, *dest*, *log_level='info'*, *error_level='error'*, *copystat=False*, *compress=False*)
copy or compress *src* into *dest*.

Args: src (str): filepath to copy. dest (str): filepath where to move the content to. log_level (str, optional): log level to use for normal operation. Defaults to

INFO

error_level (str, optional): log level to use on error. Defaults to *ERROR* copystat (bool, optional): whether or not to copy the files metadata.

Defaults to *False*.

compress (bool, optional): whether or not to compress the destination file. Defaults to *False*.

Returns: int: -1 on error None: on success

copytree(*src*, *dest*, *overwrite='no_overwrite'*, *log_level='info'*, *error_level='error'*)
An implementation of *shutil.copytree* that allows for *dest* to exist and implements different overwrite levels: - 'no_overwrite' will keep all(any) existing files in destination tree - 'overwrite_if_exists' will only overwrite destination paths that have

the same path names relative to the root of the src and destination tree

•'clobber' will replace the whole destination tree(clobber) if it exists

Args: src (str): directory path to move. dest (str): directory path where to move the content to. overwrite (str): string specifying the overwrite level. log_level (str, optional): log level to use for normal operation. Defaults to

INFO

error_level (str, optional): log level to use on error. Defaults to *ERROR*

Returns: int: -1 on error None: on success

download_file(*url*, *file_name=None*, *parent_dir=None*, *create_parent_dir=True*, *error_level='error'*, *exit_code=3*, *retry_config=None*)
Python wget. Download the filename at *url* into *file_name* and put it on *parent_dir*. On error log with the specified *error_level*, on fatal exit with *exit_code*. Execute all the above based on *retry_config* parameter.

Args: url (str): URL path where the file to be downloaded is located. file_name (str, optional): file_name where the file will be written to.

Defaults to urls' filename.

parent_dir (str, optional): directory where the downloaded file will be written to. Defaults to current working directory

create_parent_dir (bool, optional): create the parent directory if it doesn't exist. Defaults to *True*

error_level (str, optional): log level to use in case an error occurs. Defaults to *ERROR*

retry_config (dict, optional): key-value pairs to be passed to *self.retry*. Defaults to *None*

Returns: str: filename where the downloaded file was written to. unknown: on failure, *failure_status* is returned.

env = None

get_filename_from_url(*url*)
parse a filename base on an url.

Args: url (str): url to parse for the filename

Returns:

str: filename parsed from the url, or *netloc* network location part of the url.

get_output_from_command(*command*, *cwd=None*, *halt_on_failure=False*, *env=None*, *silent=False*, *log_level='info'*, *tmpfile_base_path='tmpfile'*, *return_type='output'*, *save_tmpfiles=False*, *throw_exception=False*, *fatal_exit_code=2*, *ignore_errors=False*, *success_codes=None*)
Similar to run_command, but where run_command is an os.system(command) analog, get_output_from_command is a *command* analog.

Less error checking by design, though if we figure out how to do it without borking the output, great.

TODO: binary mode? silent is kinda like that. TODO: since p.wait() can take a long time, optionally log something every N seconds? TODO: optionally only keep the first or last (N) line(s) of output? TODO: optionally only return the tmp_stdout_filename?

ignore_errors=True is for the case where a command might produce standard error output, but you don't particularly care; setting to True will cause standard error to be logged at DEBUG rather than ERROR

Args:

command (str | list): command or list of commands to execute and log.

cwd (str, optional): directory path from where to execute the command. Defaults to *None*.

halt_on_failure (bool, optional): whether or not to redefine the log level as *FATAL* on error. Defaults to False.

env (dict, optional): key-value of environment values to use to run the command. Defaults to None.

silent (bool, optional): whether or not to output the stdout of executing the command. Defaults to False.

log_level (str, optional): log level name to use on normal execution. Defaults to *INFO*.

tmpfile_base_path (str, optional): base path of the file to which the output will be writen to. Defaults to 'tmpfile'.

return_type (str, optional): if equal to 'output' then the complete output of the executed command is returned, otherwise the written filenames are returned. Defaults to 'output'.

save_tmpfiles (bool, optional): whether or not to save the temporary files created from the command output. Defaults to False.

throw_exception (bool, optional): whether or not to raise an exception if the return value of the command is not zero. Defaults to False.

fatal_exit_code (int, optional): call self.fatal if the return value of the command match this value.

ignore_errors (bool, optional): whether or not to change the log level to *ERROR* for the output of stderr. Defaults to False.

success_codes (int, optional): numeric value to compare against the command return value.

Returns: None: if the cwd is not a directory. None: on IOError. tuple: stdout and stderr filenames. str: stdout output.

is_exe (*fpath*)
Determine if fpath is a file and if it is executable.

mkdir_p (*path*, *error_level='error'*)
Create a directory if it doesn't exists. This method also logs the creation, error or current existence of the directory to be created.

Args: path (str): path of the directory to be created. error_level (str): log level name to be used in case of error.

Returns: None: for sucess. int: -1 on error

move (*src*, *dest*, *log_level='info'*, *error_level='error'*, *exit_code=-1*)
recursively move a file or directory (src) to another location (dest).

Args: src (str): file or directory path to move. dest (str): file or directory path where to move the content to. log_level (str): log level to use for normal operation. Defaults to

INFO

error_level (str): log level to use on error. Defaults to *ERROR*

Returns: int: 0 on success. -1 on error.

opened(**args*, ***kwds*)
Create a context manager to use on a with statement.

Args: file_path (str): filepath of the file to open. verbose (bool, optional): useless parameter, not used here.

Defaults to True.

open_mode (str, optional): open mode to use for openning the file. Defaults to *r*

error_level (str, optional): log level name to use on error. Defaults to *ERROR*

Yields:

tuple: (file object, error) pair. In case of error *None* is yielded as file object, together with the corresponding error. If there is no error, *None* is returned as the error.

query_env(*partial_env=None*, *replace_dict=None*, *purge_env=()*, *set_self_env=None*, *log_level='debug'*, *avoid_host_env=False*)
Environment query/generation method. The default, self.query_env(), will look for self.config['env'] and replace any special strings in there (%(PATH)s). It will then store it as self.env for speeding things up later.

If you specify partial_env, partial_env will be used instead of self.config['env'], and we don't save self.env as it's a one-off.

Args:

partial_env (dict, optional): key-value pairs of the name and value of different environment variables. Defaults to an empty dictionary.

replace_dict (dict, optional): key-value pairs to replace the old environment variables.

purge_env (list): environment names to delete from the final environment dictionary.

set_self_env (boolean, optional): whether or not the environment variables dictionary should be copied to *self*. Defaults to True.

log_level (str, optional): log level name to use on normal operation. Defaults to *DEBUG*.

avoid_host_env (boolean, optional): if set to True, we will not use any environment variables set on the host except PATH. Defaults to False.

Returns: dict: environment variables names with their values.

query_exe(*exe_name*, *exe_dict='exes'*, *default=None*, *return_type=None*, *error_level='fatal'*)
One way to work around PATH rewrites.

By default, return exe_name, and we'll fall through to searching os.environ["PATH"]. However, if self.config[exe_dict][exe_name] exists, return that. This lets us override exe paths via config file.

If we need runtime setting, we can build in self.exes support later.

Args: exe_name (str): name of the executable to search for. exe_dict(str, optional): name of the dictionary of executables

present in *self.config*. Defaults to *exes*.

default (str, optional): default name of the executable to search for. Defaults to *exe_name*.

return_type (str, optional): type to which the original return value will be turn into. Only 'list', 'string' and *None* are supported. Defaults to *None*.

error_level (str, optional): log level name to use on error.

Returns: list: in case return_type is 'list' str: in case return_type is 'string' None: in case return_type is *None* Any: if the found executable is not of type list, tuple nor str.

query_msys_path (*path*)
replaces the Windows harddrive letter path style with a linux path style, e.g. C:// –> /C/ Note: method, not used in any script.

Args: path (str?): path to convert to the linux path style.

Returns: str: in case *path* is a string. The result is the path with the new notation. type(path): *path* itself is returned in case *path* is not str type.

read_from_file (*file_path*, *verbose=True*, *open_mode='r'*, *error_level='error'*)
Use *self.opened* context manager to open a file and read its content.

Args: file_path (str): filepath of the file to read. verbose (bool, optional): whether or not to log the file content.

Defaults to True.

open_mode (str, optional): open mode to use for openning the file. Defaults to *r*

error_level (str, optional): log level name to use on error. Defaults to *ERROR*

Returns: None: on error. str: file content on success.

retry (*action*, *attempts=None*, *sleeptime=60*, *max_sleeptime=300*, *retry_exceptions=(<type 'exceptions.Exception'>,)*, *good_statuses=None*, *cleanup=None*, *error_level='error'*, *error_message='%(action)s failed after %(attempts)d tries!'*, *failure_status=-1*, *log_level='info'*, *args=()*, *kwargs={}*)
generic retry command. Ported from **'util.retry'_**

Args: action (func): callable object to retry. attempts (int, optinal): maximum number of times to call actions.

Defaults to *self.config.get('global_retries', 5)*

sleeptime (int, optional): number of seconds to wait between attempts. Defaults to 60 and doubles each retry attempt, to a maximum of 'max_sleeptime'

max_sleeptime (int, optional): maximum value of sleeptime. Defaults to 5 minutes

retry_exceptions (tuple, optional): Exceptions that should be caught. If exceptions other than those listed in 'retry_exceptions' are raised from 'action', they will be raised immediately. Defaults to (Exception)

good_statuses (object, optional): return values which, if specified, will result in retrying if the return value isn't listed. Defaults to *None*.

cleanup (func, optional): If 'cleanup' is provided and callable it will be called immediately after an Exception is caught. No arguments will be passed to it. If your cleanup function requires arguments it is recommended that you wrap it in an argumentless function. Defaults to *None*.

error_level (str, optional): log level name in case of error. Defaults to *ERROR*.

error_message (str, optional): string format to use in case none of the attempts success. Defaults to '%(action)s failed after %(attempts)d tries!'

failure_status (int, optional): flag to return in case the retries were not successfull. Defaults to -1.

log_level (str, optional): log level name to use for normal activity. Defaults to *INFO*.

args (tuple, optional): positional arguments to pass onto *action*. kwargs (dict, optional): key-value arguments to pass onto *action*.

Returns: object: return value of *action*. int: failure status in case of failure retries.

rmtree(*path*, *log_level='info'*, *error_level='error'*, *exit_code=-1*)

Delete an entire directory tree and log its result. This method also logs the platform rmtree function, its retries, errors, and current existence of the directory.

Args: path (str): path to the directory tree root to remove. log_level (str, optional): log level name to for this operation. Defaults

to *INFO*.

error_level (str, optional): log level name to use in case of error. Defaults to *ERROR*.

exit_code (int, optional): useless parameter, not use here. Defaults to -1

Returns: None: for success

run_command(*command*, *cwd=None*, *error_list=None*, *halt_on_failure=False*, *success_codes=None*, *env=None*, *partial_env=None*, *return_type='status'*, *throw_exception=False*, *output_parser=None*, *output_timeout=None*, *fatal_exit_code=2*, *error_level='error'*, ***kwargs*)

Run a command, with logging and error parsing. TODO: context_lines

error_list example: [{'regex': re.compile('^Error: LOL J/K'), level=IGNORE},

{'regex': re.compile('^Error:'), level=ERROR, contextLines='5:5'}, {'substr': 'THE WORLD IS ENDING', level=FATAL, contextLines='20:'}

] (context_lines isn't written yet)

Args:

command (str | list | tuple): command or sequence of commands to execute and log.

cwd (str, optional): directory path from where to execute the command. Defaults to *None*.

error_list (list, optional): list of errors to pass to *mozharness.base.log.OutputParser*. Defaults to *None*.

halt_on_failure (bool, optional): whether or not to redefine the log level as *FATAL* on errors. Defaults to False.

success_codes (int, optional): numeric value to compare against the command return value.

env (dict, optional): key-value of environment values to use to run the command. Defaults to None.

partial_env (dict, optional): key-value of environment values to replace from the current environment values. Defaults to None.

return_type (str, optional): if equal to 'num_errors' then the amount of errors matched by *error_list* is returned. Defaults to 'status'.

throw_exception (bool, optional): whether or not to raise an exception if the return value of the command doesn't match any of the *success_codes*. Defaults to False.

output_parser (OutputParser, optional): lets you provide an instance of your own OutputParser subclass. Defaults to *OutputParser*.

output_timeout (int): amount of seconds to wait for output before the process is killed.

fatal_exit_code (int, optional): call *self.fatal* if the return value of the command is not on in *success_codes*. Defaults to 2.

error_level (str, optional): log level name to use on error. Defaults to *ERROR*.

**kwargs: Arbitrary keyword arguments.

Returns: int: -1 on error. Any: *command* return value is returned otherwise.

script_obj = None

unpack(*filename*, *extract_to*)
This method allows us to extract a file regardless of its extension

Args: filename (str): filename of the compressed file. extract_to (str): where to extract the compressed file.

which(*program*)
OS independent implementation of Unix's which command

Args:

program (str): name or path to the program whose executable is being searched.

Returns: None: if the executable was not found. str: filepath of the executable file.

write_to_file(*file_path*, *contents*, *verbose=True*, *open_mode='w'*, *create_parent_dir=False*, *error_level='error'*)
Write *contents* to *file_path*, according to *open_mode*.

Args: file_path (str): filepath where the content will be written to. contents (str): content to write to the filepath. verbose (bool, optional): whether or not to log *contents* value.

Defaults to *True*

open_mode (str, optional): open mode to use for openning the file. Defaults to *w*

create_parent_dir (bool, optional): whether or not to create the parent directory of *file_path*

error_level (str, optional): log level to use on error. Defaults to *ERROR*

Returns: str: *file_path* on success None: on error.

mozharness.base.script.**platform_name**()

1.2.11 mozharness.base.signing module

Generic signing methods.

class mozharness.base.signing.**AndroidSigningMixin**
Bases: object

Generic Android apk signing methods.

Dependent on BaseScript.

align_apk(*unaligned_apk*, *aligned_apk*, *error_level='error'*)
Zipalign apk. Returns None on success, not None on failure.

key_passphrase = None

passphrase()

postflight_passphrase()

sign_apk(*apk*, *keystore*, *storepass*, *keypass*, *key_alias*, *remove_signature=True*, *error_list=None*, *log_level='info'*, *error_level='error'*)
Signs an apk with jarsigner.

store_passphrase = **None**

unsign_apk(*apk*, ***kwargs*)

verify_passphrases()

class mozharness.base.signing.**BaseSigningMixin**
Bases: object

Generic signing helper methods.

query_filesize(*file_path*)

query_sha512sum(*file_path*)

1.2.12 mozharness.base.transfer module

Generic ways to upload + download files.

class mozharness.base.transfer.**TransferMixin**
Bases: object

Generic transfer methods.

Dependent on BaseScript.

load_json_from_url(*url*, *timeout=30*, *log_level='debug'*)

rsync_download_directory(*ssh_key*, *ssh_user*, *remote_host*, *remote_path*, *local_path*, *rsync_options=None*, *error_level='error'*)
rsync+ssh the content of a remote directory to local_path

Returns:

None: on success -1: if local_path is not a directory -3: rsync fails to download from the remote directory

rsync_upload_directory(*local_path*, *ssh_key*, *ssh_user*, *remote_host*, *remote_path*, *rsync_options=None*, *error_level='error'*, *create_remote_directory=True*)
Create a remote directory and upload the contents of a local directory to it via rsync+ssh.

Returns:

None: on success -1: if local_path is not a directory -2: if the remote_directory cannot be created

(it only makes sense if create_remote_directory is True)

-3: rsync fails to copy to the remote directory

1.2.13 Module contents

1.3 mozharness.base.vcs package

1.3.1 Submodules

1.3.2 mozharness.base.vcs.gittool module

class mozharness.base.vcs.gittool.**GittoolParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can find the "Got revision" string from gittool.py output

got_revision = None

got_revision_exp = <_sre.SRE_Pattern object>

parse_single_line(*line*)

class mozharness.base.vcs.gittool.**GittoolVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

1.3.3 mozharness.base.vcs.hgtool module

class mozharness.base.vcs.hgtool.**HgtoolParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can find the "Got revision" string from hgtool.py output

got_revision = None

got_revision_exp = <_sre.SRE_Pattern object>

parse_single_line(*line*)

class mozharness.base.vcs.hgtool.**HgtoolVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

1.3.4 mozharness.base.vcs.mercurial module

Mercurial VCS support.

Largely copied/ported from https://hg.mozilla.org/build/tools/file/cf265ea8fb5e/lib/python/util/hg.py .

class mozharness.base.vcs.mercurial.**MercurialVCS**(*log_obj=None*, *config=None*, *vcs_config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*, object

apply_and_push(*localrepo*, *remote*, *changer*, *max_attempts=10*, *ssh_username=None*, *ssh_key=None*)

This function calls 'changer' to make changes to the repo, and tries its hardest to get them to the origin repo. 'changer' must be a callable object that receives two arguments: the directory of the local repository, and the attempt number. This function will push ALL changesets missing from remote.

cleanOutgoingRevs(*reponame*, *remote*, *username*, *sshKey*)

clone(*repo*, *dest*, *branch=None*, *revision=None*, *update_dest=True*)

Clones hg repo and places it at *dest*, replacing whatever else is there. The working copy will be empty.

If *revision* is set, only the specified revision and its ancestors will be cloned. If revision is set, branch is ignored.

If *update_dest* is set, then *dest* will be updated to *revision* if set, otherwise to *branch*, otherwise to the head of default.

common_args(*revision=None*, *branch=None*, *ssh_username=None*, *ssh_key=None*)

Fill in common hg arguments, encapsulating logic checks that depend on mercurial versions and provided arguments

ensure_repo_and_revision()

Makes sure that *dest* is has *revision* or *branch* checked out from *repo*.

Do what it takes to make that happen, including possibly clobbering dest.

get_branch_from_path(*path*)

get_branches_from_path(*path*)

get_repo_name(*repo*)

get_repo_path(*repo*)

get_revision_from_path(*path*)

Returns which revision directory *path* currently has checked out.

hg_ver()

Returns the current version of hg, as a tuple of (major, minor, build)

out(*src*, *remote*, ***kwargs*)

Check for outgoing changesets present in a repo

pull(*repo*, *dest*, *update_dest=True*, ***kwargs*)

Pulls changes from hg repo and places it in *dest*.

If *revision* is set, only the specified revision and its ancestors will be pulled.

If *update_dest* is set, then *dest* will be updated to *revision* if set, otherwise to *branch*, otherwise to the head of default.

push(*src*, *remote*, *push_new_branches=True*, ***kwargs*)

query_can_share()

share(*source*, *dest*, *branch=None*, *revision=None*)

Creates a new working directory in "dest" that shares history with "source" using Mercurial's share extension

update(*dest*, *branch=None*, *revision=None*)
Updates working copy *dest* to *branch* or *revision*. If revision is set, branch will be ignored. If neither is set then the working copy will be updated to the latest revision on the current branch. Local changes will be discarded.

mozharness.base.vcs.mercurial.**make_hg_url**(*hg_host*, *repo_path*, *protocol='http'*, *revision=None*, *filename=None*)
Helper function.

Construct a valid hg url from a base hg url (hg.mozilla.org), repo_path, revision and possible filename

1.3.5 mozharness.base.vcs.vcsbase module

Generic VCS support.

class mozharness.base.vcs.vcsbase.**MercurialScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSScript*

default_vcs = 'hg'

class mozharness.base.vcs.vcsbase.**VCSMixin**
Bases: object

Basic VCS methods that are vcs-agnostic. The vcs_class handles all the vcs-specific tasks.

query_dest(*kwargs*)

vcs_checkout(*vcs=None*, *error_level='fatal'*, ***kwargs*)
Check out a single repo.

vcs_checkout_repos(*repo_list*, *parent_dir=None*, *tag_override=None*, ***kwargs*)
Check out a list of repos.

class mozharness.base.vcs.vcsbase.**VCSScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSMixin*, *mozharness.base.script.BaseScript*

pull(*repos=None*, *parent_dir=None*)

1.3.6 mozharness.base.vcs.vcssync module

Generic VCS support.

class mozharness.base.vcs.vcssync.**VCSSyncScript**(***kwargs*)
Bases: *mozharness.base.vcs.vcsbase.VCSScript*

notify(*message=None*, *fatal=False*)
Email people in the notify_config (depending on status and failure_only)

start_time = 1440001632.71014

1.3.7 Module contents

1.4 mozharness.mozilla.building package

1.4.1 Submodules

1.4.2 mozharness.mozilla.building.buildbase module

buildbase.py.

provides a base class for fx desktop builds author: Jordan Lund

class mozharness.mozilla.building.buildbase.**BuildOptionParser**
Bases: object

bits = None

branch_cfg_file = 'builds/branch_specifics.py'

build_pool_cfg_file = 'builds/build_pool_specifics.py'

build_variants = {'api-9': 'builds/releng_sub_%s_configs/%s_api_9.py', 'api-11': 'builds/releng_sub_%s_configs/%s

config_file_search_path = ['.', '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/checko

platform = None

classmethod set_bits(*option*, *opt*, *value*, *parser*)

classmethod set_build_branch(*option*, *opt*, *value*, *parser*)

classmethod set_build_pool(*option*, *opt*, *value*, *parser*)

classmethod set_build_variant(*option*, *opt*, *value*, *parser*)
sets an extra config file.

This is done by either taking an existing filepath or by taking a valid shortname coupled with known platform/bits.

classmethod set_platform(*option*, *opt*, *value*, *parser*)

class mozharness.mozilla.building.buildbase.**BuildScript**(***kwargs*)
Bases: *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.mozilla.purge.PurgeMixin*, *mozharness.mozilla.mock.MockMixin*, mozharness.mozilla.updates.balrog.BalrogMixin, *mozharness.mozilla.signing.SigningMixin*, *mozharness.base.python.VirtualenvMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.base.transfer.TransferMixin*, *mozharness.base.python.InfluxRecordingMixin*

build()
builds application.

check_test()

checkout_sources()

clone_tools()
clones the tools repo.

generate_build_props(*console_output=True*, *halt_on_failure=False*)
sets props found from mach build and, in addition, buildid, sourcestamp, appVersion, and appName.

generate_build_stats()
grab build stats following a compile.

This action handles all statistics from a build: 'count_ctors' and then posts to graph server the results. We only post to graph server for non nightly build

multi_l10n()

package_source()
generates source archives and uploads them

postflight_build(*console_output=True*)
grabs properties from post build and calls ccache -s

preflight_build()
set up machine state for a complete build.

preflight_package_source()

query_build_env(*replace_dict=None*, ***kwargs*)

query_buildid()

query_builduid()

query_check_test_env()

query_mach_build_env(*multiLocale=None*)

query_pushdate()

query_revision(*source_path=None*)
returns the revision of the build

first will look for it in buildbot_properties and then in buildbot_config. Failing that, it will actually poll the source of the repo if it exists yet.

This method is used both to figure out what revision to check out and to figure out what revision *was* checked out.

sendchange()

update()
submit balrog update steps.

upload_files()

class mozharness.mozilla.building.buildbase.**BuildingConfig**(*config=None*, *initial_config_file=None*, *config_options=None*, *all_actions=None*, *default_actions=None*, *volatile_config=None*, *option_args=None*, *require_config_file=False*, *append_env_variables_from_configs=False*, *usage='usage: %prog [options]'*)

Bases: *mozharness.base.config.BaseConfig*

get_cfgs_from_files(*all_config_files*, *options*)
Determine the configuration from the normal options and from *–branch*, *–build-pool*, and *–custom-build-*

variant-cfg. If the files for any of the latter options are also given with *–config-file* or *–opt-config-file*, they are only parsed once.

The build pool has highest precedence, followed by branch, build variant, and any normally-specified configuration files.

class mozharness.mozilla.building.buildbase.**CheckTestCompleteParser**(***kwargs*)
Bases: *mozharness.base.log.OutputParser*

evaluate_parser()

parse_single_line(*line*)

tbpl_error_list = [{'regex': <_sre.SRE_Pattern object at 0x7fd926f5fc38>, 'level': 'RETRY'}, {'regex': <_sre.SRE_

class mozharness.mozilla.building.buildbase.**MakeUploadOutputParser**(*use_package_as_marfile=False*, *package_filename=None*, ***kwargs*)
Bases: *mozharness.base.log.OutputParser*

parse_single_line(*line*)

property_conditions = [('symbolsUrl', "m.endswith('crashreporter-symbols.zip') or m.endswith('crashreporter-sym

tbpl_error_list = [{'regex': <_sre.SRE_Pattern object at 0x7fd926f5fc38>, 'level': 'RETRY'}, {'regex': <_sre.SRE_

mozharness.mozilla.building.buildbase.**generate_build_ID**()

mozharness.mozilla.building.buildbase.**generate_build_UID**()

1.4.3 Module contents

1.5 mozharness.mozilla.l10n package

1.5.1 Submodules

1.5.2 mozharness.mozilla.l10n.locales module

Localization.

class mozharness.mozilla.l10n.locales.**GaiaLocalesMixin**
Bases: object

gaia_locale_revisions = None

pull_gaia_locale_source(*l10n_config*, *locales*, *base_dir*)

class mozharness.mozilla.l10n.locales.**LocalesMixin**(***kwargs*)
Bases: *mozharness.base.parallel.ChunkingMixin*

list_locales()
Stub action method.

parse_locales_file(*locales_file*)

pull_locale_source(*hg_l10n_base=None*, *parent_dir=None*, *vcs='hg'*)

query_abs_dirs()

query_locales()

run_compare_locales(*locale*, *halt_on_failure=False*)

1.5.3 mozharness.mozilla.l10n.multi_locale_build module

multi_locale_build.py

This should be a mostly generic multilocale build script.

class mozharness.mozilla.l10n.multi_locale_build.**MultiLocaleBuild**(*require_config_file=True*)
Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.base.vcs.vcsbase.Mercurial*

This class targets Fennec multilocale builds. We were considering this for potential Firefox desktop multilocale. Now that we have a different approach for B2G multilocale, it's most likely misnamed.

add_locales()

additional_packaging(*package_type='en-US'*, *env=None*)

backup_objdir()

build()

clobber()

config_options = [['–locale'], {'action': 'extend', 'dest': 'locales', 'type': 'string', 'help': 'Specify the locale(s) to rep

package(*package_type='en-US'*)

package_en_US()

package_multi()

preflight_package_multi()

pull_build_source()

restore_objdir()

upload_en_US()

upload_multi()

1.5.4 Module contents

1.6 mozharness.mozilla package

1.6.1 Subpackages

mozharness.mozilla.building package

Submodules

mozharness.mozilla.building.buildbase module

buildbase.py.

provides a base class for fx desktop builds author: Jordan Lund

class mozharness.mozilla.building.buildbase.**BuildOptionParser**
Bases: object

bits = None

branch_cfg_file = 'builds/branch_specifics.py'

build_pool_cfg_file = 'builds/build_pool_specifics.py'

build_variants = {'api-9': 'builds/releng_sub_%s_configs/%s_api_9.py', 'api-11': 'builds/releng_sub_%s_configs/%

config_file_search_path = ['.', '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/check

platform = None

classmethod set_bits(*option*, *opt*, *value*, *parser*)

classmethod set_build_branch(*option*, *opt*, *value*, *parser*)

classmethod set_build_pool(*option*, *opt*, *value*, *parser*)

classmethod set_build_variant(*option*, *opt*, *value*, *parser*)
sets an extra config file.

This is done by either taking an existing filepath or by taking a valid shortname coupled with known platform/bits.

classmethod set_platform(*option*, *opt*, *value*, *parser*)

class mozharness.mozilla.building.buildbase.**BuildScript**(***kwargs*)
Bases: *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.mozilla.purge.PurgeMixin*, *mozharness.mozilla.mock.MockMixin*, mozharness.mozilla.updates.balrog.BalrogMixin, *mozharness.mozilla.signing.SigningMixin*, *mozharness.base.python.VirtualenvMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.base.transfer.TransferMixin*, *mozharness.base.python.InfluxRecordingMixin*

build()
builds application.

check_test()

checkout_sources()

clone_tools()
clones the tools repo.

generate_build_props(*console_output=True*, *halt_on_failure=False*)
sets props found from mach build and, in addition, buildid, sourcestamp, appVersion, and appName.

generate_build_stats()
grab build stats following a compile.

This action handles all statistics from a build: 'count_ctors' and then posts to graph server the results. We only post to graph server for non nightly build

multi_l10n()

package_source()
generates source archives and uploads them

postflight_build(*console_output=True*)
grabs properties from post build and calls ccache -s

preflight_build()
set up machine state for a complete build.

preflight_package_source()

query_build_env(*replace_dict=None*, ***kwargs*)

query_buildid()

query_builduid()

query_check_test_env()

query_mach_build_env(*multiLocale=None*)

query_pushdate()

query_revision(*source_path=None*)
returns the revision of the build

first will look for it in buildbot_properties and then in buildbot_config. Failing that, it will actually poll the source of the repo if it exists yet.

This method is used both to figure out what revision to check out and to figure out what revision *was* checked out.

sendchange()

update()
submit balrog update steps.

upload_files()

class mozharness.mozilla.building.buildbase.**BuildingConfig**(*config=None*, *initial_config_file=None*, *config_options=None*, *all_actions=None*, *default_actions=None*, *volatile_config=None*, *option_args=None*, *require_config_file=False*, *append_env_variables_from_configs=False*, *usage='usage: %prog [options]'*)

Bases: *mozharness.base.config.BaseConfig*

get_cfgs_from_files(*all_config_files*, *options*)
Determine the configuration from the normal options and from *–branch*, *–build-pool*, and *–custom-build-variant-cfg*. If the files for any of the latter options are also given with *–config-file* or *–opt-config-file*, they are only parsed once.

The build pool has highest precedence, followed by branch, build variant, and any normally-specified configuration files.

class mozharness.mozilla.building.buildbase.**CheckTestCompleteParser**(***kwargs*)

Bases: *mozharness.base.log.OutputParser*

evaluate_parser()

parse_single_line(*line*)

tbpl_error_list = [{'regex': <_sre.SRE_Pattern object at 0x7fd926f5fc38>, 'level': 'RETRY'}, {'regex': <_sre.SRE_

class mozharness.mozilla.building.buildbase.**MakeUploadOutputParser**(*use_package_as_marfile=False*, *package_filename=None*, ***kwargs*)

Bases: *mozharness.base.log.OutputParser*

parse_single_line(*line*)

property_conditions = [('symbolsUrl', "m.endswith('crashreporter-symbols.zip') or m.endswith('crashreporter-sym

tbpl_error_list = [{'regex': <_sre.SRE_Pattern object at 0x7fd926f5fc38>, 'level': 'RETRY'}, {'regex': <_sre.SRE_

mozharness.mozilla.building.buildbase.**generate_build_ID**()

mozharness.mozilla.building.buildbase.**generate_build_UID**()

Module contents

mozharness.mozilla.l10n package

Submodules

mozharness.mozilla.l10n.locales module

Localization.

class mozharness.mozilla.l10n.locales.**GaiaLocalesMixin**
Bases: object

gaia_locale_revisions = None

pull_gaia_locale_source(*l10n_config*, *locales*, *base_dir*)

class mozharness.mozilla.l10n.locales.**LocalesMixin**(***kwargs*)
Bases: *mozharness.base.parallel.ChunkingMixin*

list_locales()
Stub action method.

parse_locales_file(*locales_file*)

pull_locale_source(*hg_l10n_base=None*, *parent_dir=None*, *vcs='hg'*)

query_abs_dirs()

query_locales()

run_compare_locales(*locale*, *halt_on_failure=False*)

mozharness.mozilla.l10n.multi_locale_build module

multi_locale_build.py

This should be a mostly generic multilocale build script.

class mozharness.mozilla.l10n.multi_locale_build.**MultiLocaleBuild**(*require_config_file=True*)
Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.base.vcs.vcsbase.Mercurial*

This class targets Fennec multilocale builds. We were considering this for potential Firefox desktop multilocale. Now that we have a different approach for B2G multilocale, it's most likely misnamed.

add_locales()

additional_packaging(*package_type='en-US'*, *env=None*)

backup_objdir()

build()

clobber()

config_options = [[['–locale'], {'action': 'extend', 'dest': 'locales', 'type': 'string', 'help': 'Specify the locale(s) to rep

package(*package_type='en-US'*)

package_en_US()

package_multi()

preflight_package_multi()

pull_build_source()

restore_objdir()

upload_en_US()

upload_multi()

Module contents

mozharness.mozilla.testing package

Submodules

mozharness.mozilla.testing.device module

Interact with a device via ADB or SUT.

This code is largely from https://hg.mozilla.org/build/tools/file/default/sut_tools

class mozharness.mozilla.testing.device.**ADBDeviceHandler**(***kwargs*)

Bases: *mozharness.mozilla.testing.device.BaseDeviceHandler*

check_device()

cleanup_device(*reboot=False*)

connect_device()

disconnect_device()

install_app(*file_path*)

ping_device(*auto_connect=False*, *silent=False*)

query_device_exe(*exe_name*)

query_device_file_exists(*file_name*)

query_device_id(*auto_connect=True*)

query_device_root(*silent=False*)

query_device_time()

reboot_device()

remove_device_root(*error_level='error'*)

remove_etc_hosts(*hosts_file='/system/etc/hosts'*)

set_device_time(*device_time=None*, *error_level='error'*)

uninstall_app(*package_name*, *package_root='/data/data'*, *error_level='error'*)

wait_for_device(*interval=60*, *max_attempts=20*)

class mozharness.mozilla.testing.device.**BaseDeviceHandler**(*log_obj=None*, *config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

add_device_flag(*flag*)

check_device()

cleanup_device(*reboot=False*)

default_port = None

device_flags = []

device_id = None

device_root = None

install_app(*file_path*)

ping_device()

query_device_id()

query_device_root()

query_download_filename(*file_id=None*)

reboot_device()

wait_for_device(*interval=60*, *max_attempts=20*)

exception mozharness.mozilla.testing.device.**DeviceException**

Bases: exceptions.Exception

class mozharness.mozilla.testing.device.**DeviceMixin**

Bases: object

BaseScript mixin, designed to interface with the device.

check_device()

cleanup_device(***kwargs*)

device_handler = None

device_root = None

install_app()

query_device_handler()

reboot_device()

class mozharness.mozilla.testing.device.**SUTDeviceHandler**(***kwargs*)

Bases: *mozharness.mozilla.testing.device.BaseDeviceHandler*

check_device()

cleanup_device(*reboot=False*)

install_app(*file_path*)

ping_device()

query_device_root(*strict=False*)

query_device_time()

query_devicemanager()

reboot_device()

remove_etc_hosts(*hosts_file='/system/etc/hosts'*)

set_device_time()

wait_for_device(*interval=60*, *max_attempts=20*)

class mozharness.mozilla.testing.device.**SUTDeviceMozdeviceMixin**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.SUTDeviceHandler*

This SUT device manager class makes calls through mozdevice (from mozbase) [1] directly rather than calling SUT tools.

[1] https://github.com/mozilla/mozbase/blob/master/mozdevice/mozdevice/devicemanagerSUT.py

dm = None

get_logcat()

query_devicemanager()

query_file(*filename*)

set_device_epoch_time(*timestamp=1440001627*)

mozharness.mozilla.testing.errors module

Mozilla error lists for running tests.

Error lists are used to parse output in mozharness.base.log.OutputParser.

Each line of output is matched against each substring or regular expression in the error list. On a match, we determine the 'level' of that line, whether IGNORE, DEBUG, INFO, WARNING, ERROR, CRITICAL, or FATAL.

mozharness.mozilla.testing.mozpool module

Interact with mozpool/lifeguard/bmm.

class mozharness.mozilla.testing.mozpool.**MozpoolMixin**
Bases: object

determine_mozpool_host(*device*)

mobile_imaging_format = 'http://mobile-imaging'

mozpool_handler = None

query_mozpool_handler(*device=None*, *mozpool_api_url=None*)

retrieve_android_device(*b2gbase*)

retrieve_b2g_device(*b2gbase*)

mozharness.mozilla.testing.talos module

run talos tests in a virtualenv

class mozharness.mozilla.testing.talos.**Talos**(***kwargs*)

Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadMi*

install and run Talos tests: https://wiki.mozilla.org/Buildbot/Talos

clone_talos()

config_options = [[['–talos-url'], {'action': 'store', 'dest': 'talos_url', 'default': 'https://hg.mozilla.org/build/talos/arc

create_virtualenv(***kwargs*)

VirtualenvMixin.create_virtualenv() assuemes we're using self.config['virtualenv_modules']. Since we are installing talos from its source, we have to wrap that method here.

download_talos_json()

postflight_create_virtualenv()

This belongs in download_and_install() but requires the virtualenv to be set up :(

The real fix here may be a –tpmanifest option for PerfConfigurator.

preflight_run_tests()

query_abs_dirs()

query_abs_pagesets_paths()

Returns a bunch of absolute pagesets directory paths. We need this to make the dir and copy the manifest to the local dir.

query_pagesets_manifest_filename()

query_pagesets_manifest_parent_path()

query_pagesets_manifest_path()

We have to copy the tp manifest from webroot to talos root when those two directories aren't the same, until bug 795172 is fixed.

Helper method to avoid hardcodes.

query_pagesets_parent_dir_path()

We have to copy the pageset into the webroot separately.

Helper method to avoid hardcodes.

query_pagesets_url()

Certain suites require external pagesets to be downloaded and extracted.

query_sps_profile_options()

query_talos_json_config()

Return the talos json config; download and read from the talos_json_url if need be.

query_talos_json_url()

Hacky, but I haven't figured out a better way to get the talos json url before we install the build.

We can't get this information after we install the build, because we have to create the virtualenv to use mozinstall, and talos_url is specified in the talos json.

query_talos_options()

query_talos_repo()

Where do we install the talos python package from? This needs to be overrideable by the talos json.

query_talos_revision()

Which talos revision do we want to use? This needs to be overrideable by the talos json.

query_tests()
Determine if we have tests to run.

Currently talos json will take precedence over config and command line options; if that's not a good default we can switch the order.

run_tests(*args=None*, ***kw*)
run Talos tests

talos_conf_path(*conf*)
return the full path for a talos .yml configuration file

talos_options(*args=None*, ***kw*)
return options to talos

class mozharness.mozilla.testing.talos.**TalosOutputParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)
Bases: *mozharness.base.log.OutputParser*

minidump_output = None

minidump_regex = <_sre.SRE_Pattern object at 0x35b9d50>

parse_single_line(*line*)
In Talos land, every line that starts with RETURN: needs to be printed with a TinderboxPrint:

worst_tbpl_status = 'SUCCESS'

mozharness.mozilla.testing.testbase module

class mozharness.mozilla.testing.testbase.**TestingMixin**(**args*, ***kwargs*)
Bases: *mozharness.base.python.VirtualenvMixin*, *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.base.python.ResourceMonitoringMixin*, *mozharness.mozilla.tooltool.TooltoolMixi*
mozharness.mozilla.testing.try_tools.TryToolsMixin

The steps to identify + download the proper bits for [browser] unit tests and Talos.

binary_path = None

default_tools_repo = 'https://hg.mozilla.org/build/tools'

download_and_extract(*target_unzip_dirs=None*, *suite_categories=None*)
download and extract test zip / download installer

download_file(**args*, ***kwargs*)
This function helps not to use download of proxied files since it does not support authenticated downloads. This could be re-factored and fixed in bug 1087664.

download_proxied_file(*url*, *file_name=None*, *parent_dir=None*, *create_parent_dir=True*, *error_level='fatal'*, *exit_code=3*)

get_test_output_parser(*suite_category*, *strict=False*, *fallback_parser_class=<class 'mozharness.mozilla.testing.unittest.DesktopUnittestOutputParser'>*, ***kwargs*)
Derive and return an appropriate output parser, either the structured output parser or a fallback based on the type of logging in use as determined by configuration.

install()

install_app(*app=None*, *target_dir=None*, *installer_path=None*)
Dependent on mozinstall

installer_path = None

installer_url = None

jsshell_url = None

minidump_stackwalk_path = None

postflight_read_buildbot_config()
Determine which files to download from the buildprops.json file created via the buildbot ScriptFactory.

postflight_run_tests()
preflight commands for all tests

preflight_download_and_extract()

preflight_install()

preflight_run_tests()
preflight commands for all tests

proxxy = None

query_build_dir_url(*file_name*)
Resolve a file name to a potential url in the build upload directory where that file can be found.

query_minidump_filename()

query_minidump_stackwalk()

query_minidump_tooltool_manifest()

query_symbols_url()

query_value(*key*)
This function allows us to check for a value in the self.tree_config first and then on self.config

structured_output(*suite_category*)
Defines whether structured logging is in use in this configuration. This may need to be replaced with data from a different config at the resolution of bug 1070041 and related bugs.

symbols_path = None

symbols_url = None

test_packages_url = None

test_url = None

test_zip_path = None

tree_config = {}

mozharness.mozilla.testing.unittest module

class mozharness.mozilla.testing.unittest.**DesktopUnittestOutputParser**(*suite_category*, ***kwargs*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can parse the number of passed/failed/todo tests from the output.

append_tinderboxprint_line(*suite_name*)

evaluate_parser(*return_code*, *success_codes=None*)

parse_single_line(*line*)

class mozharness.mozilla.testing.unittest.**EmulatorMixin**
Bases: object

Currently dependent on both TooltoolMixin and TestingMixin)

install_emulator()

install_emulator_from_tooltool(*manifest_path*, *do_unzip=True*)

class mozharness.mozilla.testing.unittest.**TestSummaryOutputParserHelper**(*regex=<_sre.SRE_Pattern object>*, ***kwargs*)
Bases: *mozharness.base.log.OutputParser*

evaluate_parser()

parse_single_line(*line*)

print_summary(*suite_name*)

mozharness.mozilla.testing.unittest.**tbox_print_summary**(*pass_count*, *fail_count*, *known_fail_count=None*, *crashed=False*, *leaked=False*)

Module contents

1.6.2 Submodules

1.6.3 mozharness.mozilla.blob_upload module

class mozharness.mozilla.blob_upload.**BlobUploadMixin**(**args*, ***kwargs*)
Bases: *mozharness.base.python.VirtualenvMixin*

Provides mechanism to automatically upload files written in MOZ_UPLOAD_DIR to the blobber upload server at the end of the running script.

This is dependent on ScriptMixin and BuildbotMixin. The testing script inheriting this class is to specify as cmdline options the <blob-upload-branch> and <blob-upload-server>

upload_blobber_files()

1.6.4 mozharness.mozilla.buildbot module

Code to tie into buildbot. Ideally this will go away if and when we retire buildbot.

class mozharness.mozilla.buildbot.**BuildbotMixin**
Bases: object

buildbot_config = None

buildbot_properties = {}

buildbot_status(*tbpl_status*, *level=None*, *set_return_code=True*)

dump_buildbot_properties(*prop_list=None*, *file_name='properties'*, *error_level='error'*)

invoke_sendchange(*downloadables=None*, *branch=None*, *username='sendchange-unittest'*, *sendchange_props=None*)
Generic sendchange, currently b2g- and unittest-specific.

query_buildbot_property(*prop_name*)

query_is_nightly()
returns whether or not the script should run as a nightly build.

First will check for 'nightly_build' in self.config and if that is not True, we will also allow buildbot_config to determine for us. Failing all of that, we default to False. Note, dependancy on buildbot_config is being deprecated. Putting everything in self.config is the preference.

read_buildbot_config()

set_buildbot_property(*prop_name*, *prop_value*, *write_to_file=False*)

tryserver_email()

worst_buildbot_status = 'SUCCESS'

1.6.5 mozharness.mozilla.gaia module

Module for performing gaia-specific tasks

class mozharness.mozilla.gaia.**GaiaMixin**
Bases: object

clone_gaia(*dest*, *repo*, *use_gaia_json=False*)
Clones an hg mirror of gaia.

repo: a dict containing 'repo_path', 'revision', and optionally 'branch' parameters

use_gaia_json: if True, the repo parameter is used to retrieve a gaia.json file from a gecko repo, which in turn is used to clone gaia; if False, repo represents a gaia repo to clone.

extract_xre(*xre_url*, *xre_path=None*, *parent_dir=None*)

make_gaia(*gaia_dir*, *xre_dir*, *debug=False*, *noftu=True*, *xre_url=None*, *build_config_path=None*)

make_node_modules()

node_setup()
Set up environment for node-based Gaia tests.

npm_error_list = [{'substr': 'command not found', 'level': 'error'}, {'substr': 'npm ERR! Error:', 'level': 'error'}]

preflight_pull()

pull(***kwargs*)
Two ways of using this function: - The user specifies –gaia-repo or in a config file - The buildbot propeties exist and we query the gaia json url

for the current gecko tree

1.6.6 mozharness.mozilla.mapper module

Support for hg/git mapper

class mozharness.mozilla.mapper.**MapperMixin**

query_mapper(*mapper_url*, *project*, *vcs*, *rev*, *require_answer=True*, *attempts=30*, *sleeptime=30*, *project_name=None*)
Returns the mapped revision for the target vcs via a mapper service

Args: mapper_url (str): base url to use for the mapper service project (str): The name of the mapper project to use for lookups vcs (str): Which vcs you want the revision for. e.g. "git" to get

the git revision given an hg revision

rev (str): The original revision you want the mapping for. require_answer (bool): Whether you require a valid answer or not.

If None is acceptable (meaning mapper doesn't know about the revision you're asking about), then set this to False. If True, then will return the revision, or cause a fatal error.

attempts (int): How many times to try to do the lookup sleeptime (int): How long to sleep between attempts project_name (str): Used for logging only to give a more

descriptive name to the project, otherwise just uses the project parameter

Returns: A revision string, or None

query_mapper_git_revision(*url*, *project*, *rev*, ***kwargs*)
Returns the git revision for the given hg revision *rev* See query_mapper docs for supported parameters and docstrings

query_mapper_hg_revision(*url*, *project*, *rev*, ***kwargs*)
Returns the hg revision for the given git revision *rev* See query_mapper docs for supported parameters and docstrings

1.6.7 mozharness.mozilla.mock module

Code to integrate with mock

class mozharness.mozilla.mock.**MockMixin**
Bases: object

Provides methods to setup and interact with mock environments. https://wiki.mozilla.org/ReleaseEngineering/Applications/Mock

This is dependent on ScriptMixin

copy_mock_files(*mock_target*, *files*)
Copy files into the mock environment *mock_target*. *files* should be an iterable of 2-tuples: (src, dst)

default_mock_target = None

delete_mock_files(*mock_target*, *files*)
Delete files from the mock environment *mock_target*. *files* should be an iterable of 2-tuples: (src, dst). Only the dst component is deleted.

disable_mock()
Restore self.run_command and self.get_output_from_command to their original versions. This is the opposite of self.enable_mock()

done_mock_setup = False

enable_mock()
Wrap self.run_command and self.get_output_from_command to run inside the mock environment given by self.config['mock_target']

get_mock_output_from_command(*mock_target*, *command*, *cwd=None*, *env=None*, ***kwargs*)
Same as ScriptMixin.get_output_from_command, except runs command inside mock environment *mock_target*.

get_mock_target()

get_output_from_command_m(**args*, ***kwargs*)
Executes self.get_mock_output_from_command if we have a mock target set, otherwise executes self.get_output_from_command.

init_mock(*mock_target*)
Initialize mock environment defined by *mock_target*

install_mock_packages(*mock_target*, *packages*)
Install *packages* into mock environment *mock_target*

mock_enabled = False

reset_mock(*mock_target=None*)
rm mock lock and reset

run_command_m(**args*, ***kwargs*)
Executes self.run_mock_command if we have a mock target set, otherwise executes self.run_command.

run_mock_command(*mock_target*, *command*, *cwd=None*, *env=None*, ***kwargs*)
Same as ScriptMixin.run_command, except runs command inside mock environment *mock_target*.

setup_mock(*mock_target=None*, *mock_packages=None*, *mock_files=None*)
Initializes and installs packages, copies files into mock environment given by configuration in self.config. The mock environment is given by self.config['mock_target'], the list of packges to install given by self.config['mock_packages'], and the list of files to copy in is self.config['mock_files'].

1.6.8 mozharness.mozilla.mozbase module

class mozharness.mozilla.mozbase.**MozbaseMixin**(**args*, ***kwargs*)
Bases: object

Automatically set virtualenv requirements to use mozbase from test package.

1.6.9 mozharness.mozilla.purge module

Purge/clobber support

class mozharness.mozilla.purge.**PurgeMixin**
Bases: object

clobber(*always_clobber_dirs=None*)
Mozilla clobberer-type clobber.

clobber_tool = '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/checkouts/latest/external_

clobberer()

default_maxage = 14

default_periodic_clobber = 168

default_skips = ['info', 'rel-*', 'tb-rel-*']

purge_builds(*basedirs=None*, *min_size=None*, *skip=None*, *max_age=None*)

purge_tool = '/home/docs/checkouts/readthedocs.org/user_builds/moz-releng-mozharness/checkouts/latest/external_to

1.6.10 mozharness.mozilla.release module

release.py

class mozharness.mozilla.release.**ReleaseMixin**

query_release_config()

release_config = {}

1.6.11 mozharness.mozilla.repo_manifest module

Module for handling repo style XML manifests

mozharness.mozilla.repo_manifest.**add_project**(*manifest*, *name*, *path*, *remote=None*, *revision=None*)
Adds a project to the manifest in place

mozharness.mozilla.repo_manifest.**cleanup**(*manifest*, *depth=0*)
Remove any empty text nodes

mozharness.mozilla.repo_manifest.**get_default**(*manifest*)

mozharness.mozilla.repo_manifest.**get_project**(*manifest*, *name=None*, *path=None*)
Gets a project node from the manifest. One of name or path must be set. If path is specified, then the project with the given path is returned, otherwise the project with the given name is returned.

mozharness.mozilla.repo_manifest.**get_project_remote_url**(*manifest*, *project*)
Gets the remote URL for the given project node. Will return the default remote if the project doesn't explicitly specify one.

mozharness.mozilla.repo_manifest.**get_project_revision**(*manifest*, *project*)
Gets the revision for the given project node. Will return the default revision if the project doesn't explicitly specify one.

mozharness.mozilla.repo_manifest.**get_remote**(*manifest*, *name*)

mozharness.mozilla.repo_manifest.**is_commitid**(*revision*)
Returns True if revision looks like a commit id i.e. 40 character string made up of 0-9a-f

mozharness.mozilla.repo_manifest.**load_manifest**(*filename*)
Loads manifest from *filename* and returns a single flattened manifest Processes any <include name="..." /> nodes recursively Removes projects referenced by <remove-project name="..." /> nodes Abort on unsupported manifest tags Returns the root node of the resulting DOM

mozharness.mozilla.repo_manifest.**map_remote**(*r*, *mappings*)
Helper function for mapping git remotes

mozharness.mozilla.repo_manifest.**remove_group**(*manifest*, *group*)
Removes all projects with groups='group'

mozharness.mozilla.repo_manifest.**remove_project**(*manifest*, *name=None*, *path=None*)
Removes a project from manifest. One of name or path must be set. If path is specified, then the project with the given path is removed, otherwise the project with the given name is removed.

mozharness.mozilla.repo_manifest.**rewrite_remotes**(*manifest*, *mapping_func*, *force_all=True*)
Rewrite manifest remotes in place Returns the same manifest, with the remotes transformed by mapping_func mapping_func should return a modified remote node, or None if no changes are required If force_all is True, then it is an error for mapping_func to return None; a ValueError is raised in this case

1.6.12 mozharness.mozilla.signing module

Mozilla-specific signing methods.

class mozharness.mozilla.signing.**MobileSigningMixin**
Bases: *mozharness.base.signing.AndroidSigningMixin*, *mozharness.mozilla.signing.SigningMixi*

verify_android_signature(*apk*, *script=None*, *key_alias='nightly'*, *tools_dir='tools/'*, *env=None*)
Runs mjessome's android signature verification script. This currently doesn't check to see if the apk exists; you may want to do that before calling the method.

class mozharness.mozilla.signing.**SigningMixin**
Bases: *mozharness.base.signing.BaseSigningMixin*

Generic signing helper methods.

query_moz_sign_cmd(*formats='gpg'*)

1.6.13 mozharness.mozilla.tooltool module

module for tooltool operations

class mozharness.mozilla.tooltool.**TooltoolMixin**
Bases: object

Mixin class for handling tooltool manifests. To use a tooltool server other than the Mozilla server, override config['tooltool_servers']. To specify a different authentication file than that used in releng automation,override config['tooltool_authentication_file']; set it to None to not pass any authentication information (OK for public files)

create_tooltool_manifest(*contents*, *path=None*)
Currently just creates a manifest, given the contents. We may want a template and individual values in the future?

tooltool_fetch(*manifest*, *bootstrap_cmd=None*, *output_dir=None*, *privileged=False*, *cache=None*)
docstring for tooltool_fetch

1.6.14 Module contents

1.7 mozharness.mozilla.testing package

1.7.1 Submodules

1.7.2 mozharness.mozilla.testing.device module

Interact with a device via ADB or SUT.

This code is largely from https://hg.mozilla.org/build/tools/file/default/sut_tools

class mozharness.mozilla.testing.device.**ADBDeviceHandler**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.BaseDeviceHandler*

check_device()

cleanup_device(*reboot=False*)

connect_device()

disconnect_device()

install_app(*file_path*)

ping_device(*auto_connect=False*, *silent=False*)

query_device_exe(*exe_name*)

query_device_file_exists(*file_name*)

query_device_id(*auto_connect=True*)

query_device_root(*silent=False*)

query_device_time()

reboot_device()

remove_device_root(*error_level='error'*)

remove_etc_hosts(*hosts_file='/system/etc/hosts'*)

set_device_time(*device_time=None*, *error_level='error'*)

uninstall_app(*package_name*, *package_root='/data/data'*, *error_level='error'*)

wait_for_device(*interval=60*, *max_attempts=20*)

class mozharness.mozilla.testing.device.**BaseDeviceHandler**(*log_obj=None*, *config=None*, *script_obj=None*)

Bases: *mozharness.base.script.ScriptMixin*, *mozharness.base.log.LogMixin*

add_device_flag(*flag*)

check_device()

cleanup_device(*reboot=False*)

default_port = None

device_flags = []

device_id = None

device_root = None

install_app(*file_path*)

ping_device()

query_device_id()

query_device_root()

query_download_filename(*file_id=None*)

reboot_device()

wait_for_device(*interval=60*, *max_attempts=20*)

exception mozharness.mozilla.testing.device.**DeviceException**

Bases: exceptions.Exception

class mozharness.mozilla.testing.device.**DeviceMixin**

Bases: object

BaseScript mixin, designed to interface with the device.

check_device()

cleanup_device(***kwargs*)

device_handler = None

device_root = None

install_app()

query_device_handler()

reboot_device()

class mozharness.mozilla.testing.device.**SUTDeviceHandler**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.BaseDeviceHandler*

check_device()

cleanup_device(*reboot=False*)

install_app(*file_path*)

ping_device()

query_device_root(*strict=False*)

query_device_time()

query_devicemanager()

reboot_device()

remove_etc_hosts(*hosts_file='/system/etc/hosts'*)

set_device_time()

wait_for_device(*interval=60*, *max_attempts=20*)

class mozharness.mozilla.testing.device.**SUTDeviceMozdeviceMixin**(***kwargs*)
Bases: *mozharness.mozilla.testing.device.SUTDeviceHandler*

This SUT device manager class makes calls through mozdevice (from mozbase) [1] directly rather than calling SUT tools.

[1] https://github.com/mozilla/mozbase/blob/master/mozdevice/mozdevice/devicemanagerSUT.py

dm = None

get_logcat()

query_devicemanager()

query_file(*filename*)

set_device_epoch_time(*timestamp=1440001627*)

1.7.3 mozharness.mozilla.testing.errors module

Mozilla error lists for running tests.

Error lists are used to parse output in mozharness.base.log.OutputParser.

Each line of output is matched against each substring or regular expression in the error list. On a match, we determine the 'level' of that line, whether IGNORE, DEBUG, INFO, WARNING, ERROR, CRITICAL, or FATAL.

1.7.4 mozharness.mozilla.testing.mozpool module

Interact with mozpool/lifeguard/bmm.

class mozharness.mozilla.testing.mozpool.**MozpoolMixin**
Bases: object

determine_mozpool_host(*device*)

mobile_imaging_format = 'http://mobile-imaging'

mozpool_handler = None

query_mozpool_handler(*device=None*, *mozpool_api_url=None*)

retrieve_android_device(*b2gbase*)

retrieve_b2g_device(*b2gbase*)

1.7.5 mozharness.mozilla.testing.talos module

run talos tests in a virtualenv

class mozharness.mozilla.testing.talos.**Talos**(***kwargs*)
Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadM*

install and run Talos tests: https://wiki.mozilla.org/Buildbot/Talos

clone_talos()

config_options = [[['–talos-url'], {'action': 'store', 'dest': 'talos_url', 'default': 'https://hg.mozilla.org/build/talos/arc

create_virtualenv(***kwargs*)
VirtualenvMixin.create_virtualenv() assuemes we're using self.config['virtualenv_modules']. Since we are installing talos from its source, we have to wrap that method here.

download_talos_json()

postflight_create_virtualenv()
This belongs in download_and_install() but requires the virtualenv to be set up :(

The real fix here may be a –tpmanifest option for PerfConfigurator.

preflight_run_tests()

query_abs_dirs()

query_abs_pagesets_paths()
Returns a bunch of absolute pagesets directory paths. We need this to make the dir and copy the manifest to the local dir.

query_pagesets_manifest_filename()

query_pagesets_manifest_parent_path()

query_pagesets_manifest_path()
We have to copy the tp manifest from webroot to talos root when those two directories aren't the same, until bug 795172 is fixed.

Helper method to avoid hardcodes.

query_pagesets_parent_dir_path()
We have to copy the pageset into the webroot separately.

Helper method to avoid hardcodes.

query_pagesets_url()
Certain suites require external pagesets to be downloaded and extracted.

query_sps_profile_options()

query_talos_json_config()
Return the talos json config; download and read from the talos_json_url if need be.

query_talos_json_url()
Hacky, but I haven't figured out a better way to get the talos json url before we install the build.

We can't get this information after we install the build, because we have to create the virtualenv to use mozinstall, and talos_url is specified in the talos json.

query_talos_options()

query_talos_repo()
Where do we install the talos python package from? This needs to be overrideable by the talos json.

query_talos_revision()
Which talos revision do we want to use? This needs to be overrideable by the talos json.

query_tests()
Determine if we have tests to run.

Currently talos json will take precedence over config and command line options; if that's not a good default we can switch the order.

run_tests(*args=None*, ***kw*)
run Talos tests

talos_conf_path(*conf*)
return the full path for a talos .yml configuration file

talos_options(*args=None*, ***kw*)
return options to talos

class mozharness.mozilla.testing.talos.**TalosOutputParser**(*config=None*, *log_obj=None*, *error_list=None*, *log_output=True*)
Bases: *mozharness.base.log.OutputParser*

minidump_output = None

minidump_regex = <_sre.SRE_Pattern object at 0x35b9d50>

parse_single_line(*line*)
In Talos land, every line that starts with RETURN: needs to be printed with a TinderboxPrint:

worst_tbpl_status = 'SUCCESS'

1.7.6 mozharness.mozilla.testing.testbase module

class mozharness.mozilla.testing.testbase.**TestingMixin**(**args*, ***kwargs*)
Bases: *mozharness.base.python.VirtualenvMixin*, *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.base.python.ResourceMonitoringMixin*, *mozharness.mozilla.tooltool.TooltoolMixi*
mozharness.mozilla.testing.try_tools.TryToolsMixin

The steps to identify + download the proper bits for [browser] unit tests and Talos.

binary_path = None

default_tools_repo = 'https://hg.mozilla.org/build/tools'

download_and_extract(*target_unzip_dirs=None*, *suite_categories=None*)
download and extract test zip / download installer

download_file(**args*, ***kwargs*)
This function helps not to use download of proxied files since it does not support authenticated downloads. This could be re-factored and fixed in bug 1087664.

download_proxied_file(*url*, *file_name=None*, *parent_dir=None*, *create_parent_dir=True*, *error_level='fatal'*, *exit_code=3*)

get_test_output_parser(*suite_category*, *strict=False*, *fallback_parser_class=<class 'mozharness.mozilla.testing.unittest.DesktopUnittestOutputParser'>*, ***kwargs*)
Derive and return an appropriate output parser, either the structured output parser or a fallback based on the type of logging in use as determined by configuration.

install()

install_app(*app=None*, *target_dir=None*, *installer_path=None*)
Dependent on mozinstall

installer_path = None

installer_url = None

jsshell_url = None

minidump_stackwalk_path = None

postflight_read_buildbot_config()
Determine which files to download from the buildprops.json file created via the buildbot ScriptFactory.

postflight_run_tests()
preflight commands for all tests

preflight_download_and_extract()

preflight_install()

preflight_run_tests()
preflight commands for all tests

proxxy = None

query_build_dir_url(*file_name*)
Resolve a file name to a potential url in the build upload directory where that file can be found.

query_minidump_filename()

query_minidump_stackwalk()

query_minidump_tooltool_manifest()

query_symbols_url()

query_value(*key*)
This function allows us to check for a value in the self.tree_config first and then on self.config

structured_output(*suite_category*)
Defines whether structured logging is in use in this configuration. This may need to be replaced with data from a different config at the resolution of bug 1070041 and related bugs.

symbols_path = None

symbols_url = None

test_packages_url = None

test_url = None

test_zip_path = None

tree_config = {}

1.7.7 mozharness.mozilla.testing.unittest module

class mozharness.mozilla.testing.unittest.**DesktopUnittestOutputParser**(*suite_category*, ***kwargs*)

Bases: *mozharness.base.log.OutputParser*

A class that extends OutputParser such that it can parse the number of passed/failed/todo tests from the output.

append_tinderboxprint_line(*suite_name*)

evaluate_parser(*return_code*, *success_codes=None*)

parse_single_line(*line*)

class mozharness.mozilla.testing.unittest.**EmulatorMixin**

Bases: object

Currently dependent on both TooltoolMixin and TestingMixin)

install_emulator()

install_emulator_from_tooltool(*manifest_path*, *do_unzip=True*)

class mozharness.mozilla.testing.unittest.**TestSummaryOutputParserHelper**(*regex=<_sre.SRE_Pattern object>*, ***kwargs*)

Bases: *mozharness.base.log.OutputParser*

evaluate_parser()

parse_single_line(*line*)

print_summary(*suite_name*)

mozharness.mozilla.testing.unittest.**tbox_print_summary**(*pass_count*, *fail_count*, *known_fail_count=None*, *crashed=False*, *leaked=False*)

1.7.8 Module contents

CHAPTER 2

scripts

2.1 android_emulator_build module

class android_emulator_build.**EmulatorBuild**(*require_config_file=False*)

Bases: *mozharness.base.script.BaseScript*, *mozharness.mozilla.purge.PurgeMixin*

adb_e(*commands*)

android_apilevel(*tag*)

apt_add_repo(*repo*)

apt_get(*pkgs*)

apt_get_dependencies()

apt_update()

build_aosp()

build_kernel()

build_orangutan_su()

bundle_avds()

bundle_emulators()

checkout_orangutan()

clone_customized_avd()

config_options = [[['–host-arch'], {'dest': 'host_arch', 'help': 'architecture of the host the emulator will run on (x86,

cpu_specific_args(*avddir*)

customize_avd()

download_aosp()

download_kernel()

download_ndk()

download_test_binaries()

emu_env()

is_arm_target()

is_armv7_target()

make_base_avd()

make_one_avd(*avdname*)

ndk_bin(*b*)

ndk_bin_dir()

ndk_cross_prefix()

ndk_sysroot()

patch_aosp()

select_android_tag(*vers*)

select_patches(*tag*)

write_registry_file(*avddir*, *avdname*)

android_emulator_build.**sniff_host_arch**()

2.2 android_emulator_unittest module

class android_emulator_unittest.**AndroidEmulatorTest**(*require_config_file=False*)

Bases: *mozharness.mozilla.blob_upload.BlobUploadMixin*, *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.mozilla.testing.unittest.Em* *mozharness.base.vcs.vcsbase.VCSMixin*, *mozharness.base.script.BaseScript*, *mozharness.mozilla.mozbase.MozbaseMixin*

app_name = None

config_options = [[['–robocop-url'], {'action': 'store', 'dest': 'robocop_url', 'default': None, 'help': 'URL to the robo

download_and_extract()

Download and extract fennec APK, tests.zip, host utils, and robocop (if required).

error_list = []

install()

Install APKs on the emulator

preflight_install()

query_abs_dirs()

run_tests()

Run the tests

setup_avds()

If tooltool cache mechanism is enabled, the cached version is used by the fetch command. If the manifest includes an "unpack" field, tooltool will unpack all compressed archives mentioned in the manifest.

start_emulator()

Starts the emulator

stop_emulator()

Report emulator health, then make sure that the emulator has been stopped

verify_emulator()

Check to see if the emulator can be contacted via adb, telnet, and sut, if configured. If any communication attempt fails, kill the emulator, re-launch, and re-check.

virtualenv_modules = []

virtualenv_requirements = []

2.3 android_panda module

class android_panda.**PandaTest**(*require_config_file=False*)

Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadMi* *mozharness.mozilla.testing.mozpool.MozpoolMixin*, *mozharness.mozilla.buildbot.BuildbotMix* *mozharness.mozilla.testing.device.SUTDeviceMozdeviceMixin*, *mozharness.mozilla.mozbase.MozbaseMixin*

close_request()

config_options = [[['–mozpool-api-url'], {'dest': 'mozpool_api_url', 'help': 'Override mozpool api url'}], [['–mozpoo

download_and_extract()

Provides the target suite categories to **TestingMixin.download_**

error_list = []

mozpool_handler = None

postflight_read_buildbot_config()

query_abs_dirs()

request_device()

run_test()

test_suites = ['mochitest', 'reftest', 'crashtest', 'jsreftest', 'robocop', 'instrumentation', 'xpcshell', 'jittest', 'cppunitt

virtualenv_modules = ['mozpoolclient']

2.4 android_panda_talos module

class android_panda_talos.**PandaTalosTest**(*require_config_file=False*)

Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadMi* *mozharness.mozilla.testing.mozpool.MozpoolMixin*, *mozharness.mozilla.buildbot.BuildbotMix* *mozharness.mozilla.testing.device.SUTDeviceMozdeviceMixin*

close_request()

config_options = [[['–mozpool-api-url'], {'dest': 'mozpool_api_url', 'help': 'Override mozpool api url'}], [['–mozpoo

download_and_extract()

error_list = []

mozpool_handler = None

postflight_read_buildbot_config()

preflight_talos(*suite_category*, *suites*)

preflight perf config etc

query_abs_dirs()

query_talos_json_config()

request_device()

run_test()

test_suites = ['talos']

virtualenv_modules = ['mozpoolclient', 'mozcrash']

2.5 b2g_build module

class b2g_build.**B2GBuild**(*require_config_file=False, config={}, all_actions=['clobber', 'checkout-sources', 'checkout-gecko', 'download-gonk', 'unpack-gonk', 'checkout-gaia', 'checkout-gaia-l10n', 'checkout-gecko-l10n', 'checkout-compare-locales', 'get-blobs', 'update-source-manifest', 'build', 'build-symbols', 'make-updates', 'build-update-testdata', 'prep-upload', 'upload', 'make-socorro-json', 'upload-source-manifest', 'submit-to-balrog'], default_actions=['checkout-sources', 'get-blobs', 'build']*)

Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.mozilla.purge.PurgeMixin*, mozharness.mozilla.building.buildb2gbase.B2GBuildBaseScript, *mozharness.mozilla.l10n.locales.GaiaLocalesMixin*, *mozharness.mozilla.signing.SigningMixi* *mozharness.mozilla.mapper.MapperMixin*, mozharness.mozilla.updates.balrog.BalrogMixin, *mozharness.base.python.VirtualenvMixin*, *mozharness.base.python.InfluxRecordingMixin*

all_actions = ['clobber', 'checkout-sources', 'checkout-gecko', 'download-gonk', 'unpack-gonk', 'checkout-gaia', 'ch

build()

build_symbols()

checkout_compare_locales()

checkout_gaia_l10n()

checkout_gecko_l10n()

checkout_sources()

clobber()

config_options = [[['--gaia-languages-file'], {'dest': 'gaia_languages_file', 'help': 'languages file for gaia multilocale p

default_actions = ['checkout-sources', 'get-blobs', 'build']

download_blobs()

generate_build_command(*target=None*)

get_blobs()

get_hg_commit_time(*repo_dir*, *rev*)

Returns the commit time for given *rev* in unix epoch time

make_socorro_json()

make_updates()

prep_upload()

query_abs_dirs()

query_application_ini()

query_b2g_version()

query_branch()

query_build_env()

query_buildid()

query_complete_mar_url()

query_device_outputdir()

query_do_translate_hg_to_git(*gecko_config_key=None*)

query_do_upload()

query_dotconfig()

query_marfile_path()

query_update_channel()

query_version()

sign_updates()

submit_to_balrog()

unpack_blobs()

update_source_manifest()

upload()

upload_source_manifest()

2.6 b2g_bumper module

b2g_bumper.py

Updates a gecko repo with up to date information from B2G repositories.

In particular, it updates gaia.json which is used by B2G desktop builds, and updates the XML manifests used by device builds.

This is to tie the external repository revisions to a visible gecko commit which appears on TBPL, so sheriffs can blame the appropriate changes.

class b2g_bumper.**B2GBumper**(*require_config_file=True*)

Bases: *mozharness.base.vcs.vcsbase.VCSScript*, *mozharness.mozilla.mapper.MapperMixin*

build_commit_message(*revision_list*, *repo_name*, *repo_url*)

bump_gaia()

check_treestatus()

checkout_gecko()

checkout_manifests()

commit_manifests()

config_options = [[['–no-write'], {'dest': 'do_write', 'action': 'store_const', 'const': False, 'help': 'disable writing in-

delete_git_ref_cache()
Used to delete the git ref cache from the file system. The cache can be used to persist git ls-remote lookup results, for example to reuse them between b2g bumper runs. Since the results are stale and do not get updated, the cache should be periodically deleted, so that the new refs can be fetched. The cache can also be used across branches/devices.

export_git_ref_cache()
This action exports the git ref cache created during this run. This is useful for sharing the cache across multiple branches (for example).

filter_groups(*device_config*, *manifest*)

filter_projects(*device_config*, *manifest*)

get_revision_list(*repo_config*, *prev_revision=None*)

hg_add(*repo_path*, *path*)
Runs 'hg add' on path

hg_commit(*repo_path*, *message*)
Commits changes in repo_path, with specified user and commit message

hg_push(*repo_path*)

import_git_ref_cache()
This action imports the git ref cache created during a previous run. This is useful for sharing the cache across multiple branches (for example).

map_remotes(*manifest*)

massage_manifests()
For each device in config['devices'], we'll strip projects mentioned in 'ignore_projects', or that have group attribute mentioned in 'filter_groups'. We'll also map remote urls Finally, we'll resolve absolute refs for projects that aren't fully specified.

push()

push_loop()

query_abs_dirs()

query_devices()

query_gaia_git_rev()
Returns (and caches) the git revision for gaia corresponding to the latest hg revision on our branch.

query_manifest(*device_name*)

query_manifest_path(*device*)

query_treestatus()
Return True if we can land based on treestatus

resolve_git_ref(*remote_url*, *revision*)

resolve_refs(*manifest*)

update_gaia_json(*path*, *hg_revision*, *hg_repo_path*, *git_revision*, *git_repo*)
Update path with repo_path + revision.

If the revision hasn't changed, don't do anything. If the repo_path changes or the current json is invalid, error but don't fail.

2.7 b2g_desktop_multilocale module

class b2g_desktop_multilocale.**B2gMultilocale**(*require_config_file=False*)
Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.base.script.BaseScript*, *mozharness.base.vcs.vcsbase.VCSMixin*, *mozharness.mozilla.l10n.locales.GaiaLocalesMixin*

This is a helper script that requires MercurialBuildFactory logic to work. We may eventually make this a standalone script.

We could inherit MercurialScript instead of BaseScript + VCSMixin

build()
Do the multilocale portion of the build + packaging.

config_options = [[['–locale'], {'action': 'extend', 'dest': 'locales', 'type': 'string', 'help': 'Specify the locale(s) to rep

pull()
Clone gaia and gecko locale repos

query_abs_dirs()

2.8 b2g_desktop_unittest module

class b2g_desktop_unittest.**B2GDesktopTest**(*options=[]*, *require_config_file=False*)
Bases: *mozharness.mozilla.blob_upload.BlobUploadMixin*, *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialS*

config_options = [[['–type'], {'action': 'store', 'dest': 'test_type', 'default': 'browser', 'help': 'The type of tests to ru

download_and_extract()

error_list = [{'substr': 'FAILED (errors=', 'level': 'error'}, {'substr': 'Could not successfully complete transport of r

preflight_run_tests()

query_abs_dirs()

run_tests()
Run the tests

test_suites = ('mochitest', 'reftest')

2.9 b2g_emulator_unittest module

class b2g_emulator_unittest.**B2GEmulatorTest**(*require_config_file=False*)
Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.VCSMixin*, *mozharness.base.script.BaseScript*, *mozharness.mozilla.blob_upload.BlobUploadMixin*

config_options = [[['–type'], {'action': 'store', 'dest': 'test_type', 'default': 'browser', 'help': 'The type of tests to ru

download_and_extract()

error_list = [{'substr': 'FAILED (errors=', 'level': 'error'}, {'substr': 'Could not successfully complete transport of r

install()

preflight_run_tests()

query_abs_dirs()

run_tests()
Run the tests

test_suites = ('jsreftest', 'reftest', 'mochitest', 'mochitest-chrome', 'xpcshell', 'crashtest', 'cppunittest')

2.10 bouncer_submitter module

class bouncer_submitter.**BouncerSubmitter**(*require_config_file=True*)

__init__(*require_config_file=True*)

__module__ = 'bouncer_submitter'

_pre_config_lock(*rw_config*)

config_options = [[['–repo'], {'dest': 'repo', 'help': 'Specify source repo, e.g. releases/mozilla-beta'}], [['–revision'], {

download_shipped_locales()

load_shipped_locales()

need_shipped_locales()

query_shipped_locales_path()

submit()

submit_partials()

2.11 bump_gaia_json module

2.12 configtest module

configtest.py

Verify the .json and .py files in the configs/ directory are well-formed. Further tests to verify validity would be desirable.

This is also a good example script to look at to understand mozharness.

class configtest.**ConfigTest**(*require_config_file=False*)
Bases: *mozharness.base.script.BaseScript*

config_options = [[['–test-file'], {'action': 'extend', 'dest': 'test_files', 'help': 'Specify which config files to test'}]]

list_config_files()
Non-default action that is mainly here to demonstrate how non-default actions work in a mozharness script.

query_config_files()
This query method, much like others, caches its runtime settings in self.VAR so we don't have to figure out config_files multiple times.

test_json_configs()
Currently only "is this well-formed json?"

test_python_configs()
Currently only "will this give me a config dictionary?"

2.13 desktop_l10n module

desktop_l10n.py

This script manages Desktop repacks for nightly builds.

class desktop_l10n.**DesktopSingleLocale**(*require_config_file=True*)

Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.mozilla.release.ReleaseMix*
mozharness.mozilla.mock.MockMixin, *mozharness.mozilla.buildbot.BuildbotMixin*,
mozharness.base.vcs.vcsbase.VCSMixin, *mozharness.mozilla.signing.SigningMixin*,
mozharness.mozilla.purge.PurgeMixin, *mozharness.base.script.BaseScript*,
mozharness.mozilla.updates.balrog.BalrogMixin, mozharness.mozilla.mar.MarMixin,
mozharness.base.python.VirtualenvMixin, *mozharness.base.transfer.TransferMixin*

Manages desktop repacks

clobber()

config_options = [[['–balrog-config'], {'action': 'extend', 'dest': 'config_files', 'type': 'string', 'help': 'Specify the bal

funsize_props()
Set buildbot properties required to trigger funsize tasks responsible to generate partial updates for successfully generated locales

get_upload_files(*locale*)

make_installers(*locale*)
wrapper for make installers-(locale)

make_unpack_en_US()
wrapper for make unpack

make_upload(*locale*)
wrapper for make upload command

make_wget_en_US()
wrapper for make wget-en-US

pull()
pulls source code

query_abs_dirs()

query_bootstrap_env()
returns the env for repacks

query_l10n_env()

query_pushdate()

query_version()
Gets the version from the objdir. Only valid after setup is run.

repack()
creates the repacks and udpates

repack_locale(*locale*)
wraps the logic for comapare locale, make installers and generate complete updates.

setup()
setup step

submit_repack_to_balrog(*locale*)
submit a single locale to balrog

submit_to_balrog()
submit to barlog

summary()
generates a summary

taskcluster_upload()

2.14 desktop_unittest module

desktop_unittest.py The goal of this is to extract desktop unittesting from buildbot's factory.py

author: Jordan Lund

class desktop_unittest.**DesktopUnittest**(*require_config_file=True*)
Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadM* *mozharness.mozilla.mozbase.MozbaseMixin*, mozharness.mozilla.testing.codecoverage.CodeCov

config_options = [[['–mochitest-suite'], {'action': 'extend', 'dest': 'specified_mochitest_suites', 'type': 'string', 'help'

download_and_extract()
download and extract test zip / download installer optimizes which subfolders to extract from tests zip

get_webapprt_path(*res_dir*, *mochitest_dir*)
Get the path to the webapp runtime binary. On Mac, we copy the stub from the resources dir to the test app bundle, since we have to run it from the executable directory of a bundle in order for its windows to appear. Ideally, the build system would do this for us at build time, and we should find a way for it to do that.

preflight_cppunittest(*suites*)

preflight_mozmill(*suites*)

preflight_xpcshell(*suites*)

query_abs_app_dir()
We can't set this in advance, because OSX install directories change depending on branding and opt/debug.

query_abs_dirs()

query_abs_res_dir()
The directory containing resources like plugins and extensions. On OSX this is Contents/Resources, on all other platforms its the same as the app dir.

As with the app dir, we can't set this in advance, because OSX install directories change depending on branding and opt/debug.

run_tests()

2.15 fx_desktop_build module

fx_desktop_build.py.

script harness to build nightly firefox within Mozilla's build environment and developer machines alike

author: Jordan Lund

class fx_desktop_build.**FxDesktopBuild**
Bases: *mozharness.mozilla.building.buildbase.BuildScript*, object

query_abs_dirs()

2.16 gaia_build_integration module

class gaia_build_integration.**GaiaBuildIntegrationTest**(*require_config_file=False*)
Bases: mozharness.mozilla.testing.gaia_test.GaiaTest

run_tests()
Run the integration test suite.

2.17 gaia_integration module

class gaia_integration.**GaiaIntegrationTest**(*require_config_file=False*)
Bases: mozharness.mozilla.testing.gaia_test.GaiaTest

run_tests()
Run the integration test suite.

2.18 gaia_unit module

class gaia_unit.**GaiaUnitTest**(*require_config_file=False*)
Bases: mozharness.mozilla.testing.gaia_test.GaiaTest

pull(***kwargs*)

run_tests()
Run the unit test suite.

2.19 marionette module

class marionette.**MarionetteTest**(*require_config_file=False*)
Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadM* *mozharness.base.transfer.TransferMixin*, *mozharness.mozilla.gaia.GaiaMixin*

config_options = [[['–application'], {'action': 'store', 'dest': 'application', 'default': None, 'help': 'application name

download_and_extract()

error_list = [{'substr': 'FAILED (errors=', 'level': 'warning'}, {'substr': 'Could not successfully complete transport

install()

preflight_run_marionette()
preflight commands for all tests

pull(***kwargs*)

query_abs_dirs()

repos = []

run_marionette()
Run the Marionette tests

2.20 mobile_l10n module

mobile_l10n.py

This currently supports nightly and release single locale repacks for Android. This also creates nightly updates.

class mobile_l10n.**MobileSingleLocale**(*require_config_file=True*)
Bases: *mozharness.mozilla.mock.MockMixin*, *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.mozilla.release.ReleaseMixin*, *mozharness.mozilla.signing.MobileSigningMixin*, *mozharness.base.transfer.TransferMixin*, *mozharness.mozilla.tooltool.TooltoolMixin*, *mozharness.mozilla.buildbot.BuildbotMixin*, *mozharness.mozilla.purge.PurgeMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, mozharness.mozilla.updates.balrog.BalrogMi

add_failure(*locale*, *message*, ***kwargs*)

checkout_tools()

clobber()

config_options = [[['–locale'], {'action': 'extend', 'dest': 'locales', 'type': 'string', 'help': 'Specify the locale(s) to sig

pull()

query_abs_dirs()

query_apkfile_path(*locale*)

query_base_package_name()
Get the package name from the objdir. Only valid after setup is run.

query_buildid()
Get buildid from the objdir. Only valid after setup is run.

query_is_release()

query_repack_env()

query_revision()
Get revision from the objdir. Only valid after setup is run.

query_upload_env()

query_upload_url(*locale*)

query_version()
Get the package name from the objdir. Only valid after setup is run.

repack()

setup()

submit_to_balrog()

summary()

upload_repacks()

2.21 mobile_partner_repack module

mobile_partner_repack.py

class mobile_partner_repack.**MobilePartnerRepack**(*require_config_file=True*)
Bases: *mozharness.mozilla.l10n.locales.LocalesMixin*, *mozharness.mozilla.release.ReleaseMix*
mozharness.mozilla.signing.MobileSigningMixin, *mozharness.base.transfer.TransferMixin*,
mozharness.base.vcs.vcsbase.MercurialScript

add_failure(*platform*, *locale*, ***kwargs*)

config_options = [[['–locale'], {'action': 'extend', 'dest': 'locales', 'type': 'string', 'help': 'Specify the locale(s) to rep

download()

preflight_sign()

pull()

query_failure(*platform*, *locale*)

repack()

sign()

upload_signed_bits()

upload_unsigned_bits()

2.22 multil10n module

multil10n.py

2.23 sourcetool module

sourcetool.py

Port of tools/buildfarm/utils/hgtool.py.

TODO: sourcetool.py currently ignores work_dir completely. Maybe we should use it instead of dest ? Maybe I need to rethink work_dir?

class sourcetool.**SourceTool**(*require_config_file=False*)
Bases: *mozharness.base.script.BaseScript*

config_options = [[['–rev', '-r'], {'action': 'store', 'dest': 'vcs_revision', 'default': None, 'help': 'Specify which revisi

source()

2.24 spidermonkey_build module

class spidermonkey_build.**SpidermonkeyBuild**
Bases: *mozharness.mozilla.mock.MockMixin*, *mozharness.mozilla.purge.PurgeMixin*,
mozharness.base.script.BaseScript, *mozharness.base.vcs.vcsbase.VCSMixin*,
mozharness.mozilla.buildbot.BuildbotMixin, *mozharness.mozilla.tooltool.TooltoolMixin*,
mozharness.base.transfer.TransferMixin

build_shell()

check_expectations()

checkout_source()

checkout_tools()

clobber_analysis()

clobber_shell()

collect_analysis_output()

config_options = [[['–repo'], {'dest': 'repo', 'help': 'which gecko repo to get spidermonkey from'}], [['–source'], {'de

configure_shell()

do_checkout_source()

get_blobs()

purge()

query_abs_dirs()

query_branch()

query_buildid()

query_compiler_manifest()

query_do_upload()

query_product()

query_repo()

query_revision()

query_sixgill_manifest()

query_target()

query_upload_path()

query_upload_remote_basepath()

query_upload_remote_baseuri()

query_upload_ssh_key()

query_upload_ssh_server()

query_upload_ssh_user()

run_analysis()

setup_analysis()

upload_analysis()

spidermonkey_build.**requires**(**queries*)

Wrapper for detecting problems where some bit of information required by the wrapped step is unavailable. Use it put prepending @requires("foo"), which will check whether self.query_foo() returns something useful.

2.25 talos_script module

talos

2.26 web_platform_tests module

class web_platform_tests.**WebPlatformTest**(*require_config_file=True*)

Bases: *mozharness.mozilla.testing.testbase.TestingMixin*, *mozharness.base.vcs.vcsbase.MercurialScript*, *mozharness.mozilla.blob_upload.BlobUploadMi*

config_options = [[['–test-type'], {'action': 'extend', 'dest': 'test_type', 'help': 'Specify the test types to run.'}], [['–to

download_and_extract()

query_abs_app_dir()

We can't set this in advance, because OSX install directories change depending on branding and opt/debug.

query_abs_dirs()

run_tests()

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

a

b

c

d

f

g

m

s

t

w

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W