## NAME

ExtUtils::Install - install files from here to there

## SYNOPSIS

```
use ExtUtils::Install;

install({ 'blib/lib' => 'some/install/dir' } );

uninstall($packlist);

pm_to_blib({ 'lib/Foo/Bar.pm' => 'blib/lib/Foo/Bar.pm' });
```

## VERSION

1.55

## DESCRIPTION

Handles the installing and uninstalling of perl modules, scripts, man pages, etc...

Both install() and uninstall() are specific to the way ExtUtils::MakeMaker handles the installation and deinstallation of perl modules. They are not designed as general purpose tools.

On some operating systems such as Win32 installation may not be possible until after a reboot has occured. This can have varying consequences: removing an old DLL does not impact programs using the new one, but if a new DLL cannot be installed properly until reboot then anything depending on it must wait. The package variable

```
$ExtUtils::Install::MUST_REBOOT
```

is used to store this status.

If this variable is true then such an operation has occured and anything depending on this module cannot proceed until a reboot has occured.

If this value is defined but false then such an operation has ocurred, but should not impact later operations.

_chmod($$;$)

Wrapper to chmod() for debugging and error trapping.

_warnonce(@)

Warns about something only once.

_choke(@)

Dies with a special message.

_move_file_at_boot( $file, $target, $moan )

OS-Specific, Win32/Cygwin

Schedules a file to be moved/renamed/deleted at next boot. $file should be a filespec of an existing file $target should be a ref to an array if the file is to be deleted otherwise it should be a filespec for a rename. If the file is existing it will be replaced.

Sets $MUST_REBOOT to 0 to indicate a deletion operation has occured and sets it to 1 to indicate that a move operation has been requested.

returns 1 on success, on failure if $moan is false errors are fatal. If $moan is true then returns 0 on error and warns instead of dies.

_unlink_or_rename( $file, $tryhard, $installing )

OS-Specific, Win32/Cygwin

Tries to get a file out of the way by unlinking it or renaming it. On some OS'es (Win32 based) DLL files can end up locked such that they can be renamed but not deleted. Likewise sometimes a file can be locked such that it cant even be renamed or changed except at reboot. To handle these cases this routine finds a tempfile name that it can either rename the file out of the way or use as a proxy for the install so that the rename can happen later (at reboot).

```
$file : the file to remove.
$tryhard : should advanced tricks be used for deletion
$installing : we are not merely deleting but we want to overwrite
```

When $tryhard is not true if the unlink fails its fatal. When $tryhard is true then the file is attempted to be renamed. The renamed file is then scheduled for deletion. If the rename fails then $installing governs what happens. If it is false the failure is fatal. If it is true then an attempt is made to schedule installation at boot using a temporary file to hold the new file. If this fails then a fatal error is thrown, if it succeeds it returns the temporary file name (which will be a derivative of the original in the same directory) so that the caller can use it to install under. In all other cases of success returns $file. On failure throws a fatal error.

## Functions

_get_install_skip

Handles loading the INSTALL.SKIP file. Returns an array of patterns to use.

_have_write_access

Abstract a -w check that tries to use POSIX::access() if possible.

_can_write_dir($dir)

Checks whether a given directory is writable, taking account the possibility that the directory might not exist and would have to be created first.

Returns a list, containing: `($writable, $determined_by, @create)`

`$writable` says whether whether the directory is (hypothetically) writable

`$determined_by` is the directory the status was determined from. It will be either the `$dir`, or one of its parents.

`@create` is a list of directories that would probably have to be created to make the requested directory. It may not actually be correct on relative paths with `..` in them. But for our purposes it should work ok

_mkpath($dir,$show,$mode,$verbose,$dry_run)

Wrapper around File::Path::mkpath() to handle errors.

If $verbose is true and >1 then additional diagnostics will be produced, also this will force $show to true.

If $dry_run is true then the directory will not be created but a check will be made to see whether it would be possible to write to the directory, or that it would be possible to create the directory.

If $dry_run is not true dies if the directory can not be created or is not writable.

_copy($from,$to,$verbose,$dry_run)

Wrapper around File::Copy::copy to handle errors.

If $verbose is true and >1 then additional diagnostics will be emitted.

If $dry_run is true then the copy will not actually occur.

Dies if the copy fails.

_chdir($from)

Wrapper around chdir to catch errors.

If not called in void context returns the cwd from before the chdir.

dies on error.

**install**

```
# deprecated forms
install(\%from_to);
install(\%from_to, $verbose, $dry_run, $uninstall_shadows,
            $skip, $always_copy, \%result);

# recommended form as of 1.47
install([
    from_to => \%from_to,
    verbose => 1,
    dry_run => 0,
    uninstall_shadows => 1,
    skip => undef,
    always_copy => 1,
    result => \%install_results,
]);
```

Copies each directory tree of %from_to to its corresponding value preserving timestamps and permissions.

There are two keys with a special meaning in the hash: "read" and "write". These contain packlist files. After the copying is done, install() will write the list of target files to $from_to{write}. If $from_to{read} is given the contents of this file will be merged into the written file. The read and the written file may be identical, but on AFS it is quite likely that people are installing to a different directory than the one where the files later appear.

If $verbose is true, will print out each file removed. Default is false. This is "make install VERBINST=1". $verbose values going up to 5 show increasingly more diagnostics output.

If $dry_run is true it will only print what it was going to do without actually doing it. Default is false.

If $uninstall_shadows is true any differing versions throughout @INC will be uninstalled. This is "make install UNINST=1"

As of 1.37_02 install() supports the use of a list of patterns to filter out files that shouldn't be installed. If $skip is omitted or undefined then install will try to read the list from INSTALL.SKIP in the CWD. This file is a list of regular expressions and is just like the MANIFEST.SKIP file used by *ExtUtils::Manifest*.

A default site INSTALL.SKIP may be provided by setting then environment variable EU_INSTALL_SITE_SKIPFILE, this will only be used when there isn't a distribution specific INSTALL.SKIP. If the environment variable EU_INSTALL_IGNORE_SKIP is true then no install file filtering will be performed.

If $skip is undefined then the skip file will be autodetected and used if it is found. If $skip is a reference to an array then it is assumed the array contains the list of patterns, if $skip is a true non reference it is assumed to be the filename holding the list of patterns, any other value of $skip is taken to mean that no install filtering should occur.

**Changes As of Version 1.47**

As of version 1.47 the following additions were made to the install interface. Note that the new argument style and use of the %result hash is recommended.

The $always_copy parameter which when true causes files to be updated regardles as to whether they have changed, if it is defined but false then copies are made only if the files have changed, if it is undefined then the value of the environment variable EU_INSTALL_ALWAYS_COPY is used as default.

The %result hash will be populated with the various keys/subhashes reflecting the install. Currently these keys and their structure are:

```
install             => { $target    => $source },
install_fail        => { $target    => $source },
install_unchanged   => { $target    => $source },

install_filtered    => { $source    => $pattern },

uninstall           => { $uninstalled => $source },
uninstall_fail      => { $uninstalled => $source },
```

where `$source` is the filespec of the file being installed. `$target` is where it is being installed to, and `$uninstalled` is any shadow file that is in `@INC` or `$ENV{PERL5LIB}` or other standard locations, and `$pattern` is the pattern that caused a source file to be skipped. In future more keys will be added, such as to show created directories, however this requires changes in other modules and must therefore wait.

These keys will be populated before any exceptions are thrown should there be an error.

Note that all updates of the %result are additive, the hash will not be cleared before use, thus allowing status results of many installs to be easily aggregated.

**NEW ARGUMENT STYLE**

If there is only one argument and it is a reference to an array then the array is assumed to contain a list of key-value pairs specifying the options. In this case the option "from_to" is mandatory. This style means that you dont have to supply a cryptic list of arguments and can use a self documenting argument list that is easier to understand.

This is now the recommended interface to install().

**RETURN**

If all actions were successful install will return a hashref of the results as described above for the $result parameter. If any action is a failure then install will die, therefore it is recommended to pass in the $result parameter instead of using the return value. If the result parameter is provided then the returned hashref will be the passed in hashref.

_do_cleanup

Standardize finish event for after another instruction has occured. Handles converting $MUST_REBOOT to a die for instance.

install_rooted_file( $file )

Returns $file, or catfile($INSTALL_ROOT,$file) if $INSTALL_ROOT is defined.

install_rooted_dir( $dir )

Returns $dir, or catdir($INSTALL_ROOT,$dir) if $INSTALL_ROOT is defined.

forceunlink( $file, $tryhard )

Tries to delete a file. If $tryhard is true then we will use whatever devious tricks we can to delete the file. Currently this only applies to Win32 in that it will try to use Win32API::File to schedule a delete at reboot. A wrapper for _unlink_or_rename().

directory_not_empty( $dir )

Returns 1 if there is an .exists file somewhere in a directory tree. Returns 0 if there is not.

**install_default** *DISCOURAGED*

```
install_default();
install_default($fullext);
```

Calls install() with arguments to copy a module from blib/ to the default site installation location.

$fullext is the name of the module converted to a directory (ie. Foo::Bar would be Foo/Bar). If $fullext is not specified, it will attempt to read it from @ARGV.

This is primarily useful for install scripts.

**NOTE** This function is not really useful because of the hard-coded install location with no way to control site vs core vs vendor directories and the strange way in which the module name is given. Consider its use discouraged.

**uninstall**

```
uninstall($packlist_file);
uninstall($packlist_file, $verbose, $dont_execute);
```

Removes the files listed in a $packlist_file.

If $verbose is true, will print out each file removed. Default is false.

If $dont_execute is true it will only print what it was going to do without actually doing it. Default is false.

inc_uninstall($filepath,$libdir,$verbose,$dry_run,$ignore,$results)

Remove shadowed files. If $ignore is true then it is assumed to hold a filename to ignore. This is used to prevent spurious warnings from occurring when doing an install at reboot.

We now only die when failing to remove a file that has precedence over our own, when our install has precedence we only warn.

$results is assumed to contain a hashref which will have the keys 'uninstall' and 'uninstall_fail' populated with keys for the files removed and values of the source files they would shadow.

run_filter($cmd,$src,$dest)

Filter $src using $cmd into $dest.

**pm_to_blib**

```
pm_to_blib(\%from_to, $autosplit_dir);
pm_to_blib(\%from_to, $autosplit_dir, $filter_cmd);
```

Copies each key of %from_to to its corresponding value efficiently. Filenames with the extension .pm are autosplit into the $autosplit_dir. Any destination directories are created.

$filter_cmd is an optional shell command to run each .pm file through prior to splitting and copying. Input is the contents of the module, output the new module contents.

You can have an environment variable PERL_INSTALL_ROOT set which will be prepended as a directory to each installed file (and directory).

_autosplit

From 1.0307 back, AutoSplit will sometimes leave an open filehandle to the file being split. This causes problems on systems with mandatory locking (ie. Windows). So we wrap it and close the filehandle.

_invokant

Does a heuristic on the stack to see who called us for more intelligent error messages. Currently assumes we will be called only by Module::Build or by ExtUtils::MakeMaker.

## ENVIRONMENT

### PERL_INSTALL_ROOT

Will be prepended to each install path.

### EU_INSTALL_IGNORE_SKIP

Will prevent the automatic use of INSTALL.SKIP as the install skip file.

### EU_INSTALL_SITE_SKIPFILE

If there is no INSTALL.SKIP file in the make directory then this value can be used to provide a default.

### EU_INSTALL_ALWAYS_COPY

If this environment variable is true then normal install processes will always overwrite older identical files during the install process.

Note that the alias EU_ALWAYS_COPY will be supported if EU_INSTALL_ALWAYS_COPY is not defined until at least the 1.50 release. Please ensure you use the correct EU_INSTALL_ALWAYS_COPY.

## AUTHOR

Original author lost in the mists of time. Probably the same as Makemaker.

Production release currently maintained by demerphq `yves at cpan.org`, extensive changes by Michael G. Schwern.

Send bug reports via http://rt.cpan.org/. Please send your generated Makefile along with your report.

## LICENSE