NAME

debhelper - Version number cheatsheet and more

INTRODUCTION

This is a guide detailing some common packaging scenarios, and the version of debhelper required for them. Contributions are welcomed, especially for any packaging scenarios not already discussed here.

RECOMMENDED DEBHELPER VERSIONS

tl;dr: Use debhelper `10`.

Depending on how far back packages should be backportable, different versions of debheler make sense. With `10` *jessie-backports* and *stretch* are covered, with `11` *stretch-backports* and testing/unstable are fine.

For older releases, there are some corner cases to consider:

If `Module::Build::Tiny` is used, debhelper `9.20140227~` is required.

For packages which install non-binary files into `$Config{vendorarch}`, require `9.20140809~` in order to get the *perlapi-** dependency ; cf. #750017.

For anything else debhelper `9` is fine.

dh-make-perl(1) should be able to set the right debhelper dependency automatically.

Most Common Scenarios

Forcing Special Tests

Often, Perl packages will have special tests designated for certain environments only. It is often quite beneficial for us to run these tests during build-time, so we prefer to add the tests (if packaged) to Build-Depends-Indep.

Usually there is an environment variable, like `AUTOMATED_TESTING`, that will run these tests, however. You could enable this flag (and thus the tests) using:

```
override_dh_auto_test:
	AUTOMATED_TESTING=1 dh_auto_test
```

Sometimes tests need X server in order to run. In this case the xvfb(1) and the xauth(1) packages should be declared as `Build-Depends`. debhelper overrides should be:

```
override_dh_auto_test:
	xvfb-run -a dh_auto_test
```

dh --with bash-completion

To install `bash_completion` snippets you may use:

```
%:
	dh $@ --with bash-completion
```

Short debian/rules Format

Older versions of the rules file were pretty long and repetitive. To get most of the same features by default, all you have to do is:

```
%:
	dh $@
```

Note that the latest version of dh-make-perl does this automatically.

Occasionally Useful

Note on Paths

In order to save on typing, be a bit lazy and improve a little bit on maintainability and readability of the file, the following variables are typically placed at the top of your *debian/rules* file when there are some file operations involved.

```
PACKAGE = $(shell dh_listpackages)
TMP     = $(CURDIR)/debian/$(PACKAGE)
```

If the source package contains more than one binary packages `dh_listpackages` returns all of them, so you probably want to use:

```
PACKAGE = $(firstword $(shell dh_listpackages))
TMP     = $(CURDIR)/debian/$(PACKAGE)
```

Then any of your paths become pretty trivial, you can specify them as: **$(TMP)/usr/share/path/to/file**. Otherwise, you can accomplish the same thing with: **$(CURDIR)/debian/package-name/usr/share/path/to/file**

If you need the perl library path, which is variable since 5.20, you can use

```
ARCHLIB := $(shell perl -MConfig -e 'print $$Config{vendorarch}')
```

and later e.g. **$(TMP)/$(ARCHLIB)/...**.

Removing Something

A File

If for some reason you're not able to distribute a file, then you'll need to repack it. But if it's just causing issues like warnings with lintian or you otherwise don't want it installed, then you can remove it during build time by adding this override:

```
override_dh_auto_install:
	dh_auto_install
	$(RM) --verbose $(TMP)/usr/share/perl5/Data/Format/._HTML.pm
```

A Directory

Sometimes you want to remove an entire directory post-install. You can do that too, but it usually helps to have some extra tests so that the 'rm' operation doesn't make too much noise.

```
override_dh_auto_install:
	dh_auto_install
```

```
	rmdir --ignore-fail-on-non-empty --parents --verbose $(TMP)/$(ARCHLIB)
```

This removes empty directories, ensuring you don't accidentally destroy things that should be installed. If the package is installing files that you don't need, remove the files first before removing the empty directory.

```
override_dh_auto_install:
	dh_auto_install
	$(RM) --verbose $(TMP)/usr/share/perl5/._HTML.pm
	rmdir --ignore-fail-on-non-empty --parents --verbose $(TMP)/usr/share/perl5
```

If you're really sure, you could also do:

```
override_dh_auto_install:
	dh_auto_install
	$(RM) -rv $(TMP)/usr/share/perl5
```

But this might be dangerous, so it's not recommended.

Fixing Permissions

Sometimes debhelper's fix permissions stage (dh_fixperms) sets the wrong permissions for a given file. You can use an override to fix this, like so:

```
override_dh_fixperms:
	dh_fixperms
	chmod 644 $(TMP)/usr/share/path/to/file
```

Fixing Interpreter Shebang Lines

Sometimes examples are Perl scripts that do not use the absolute path of the Perl interpreter in the shebang line. In these cases, they should be rewritten to */usr/bin/perl*:

```
override_dh_installexamples:
	dh_installexamples
	sed -i '1s|^#!perl|#!/usr/bin/perl|' $(TMP)/usr/share/doc/$(PACKAGE)/examples/*
```

Another elegant method for achieving substitution of the shebang from */usr/local/bin/perl* to */usr/bin/perl* would be:

```
override_dh_installexamples:
	dh_installexamples
	find $(TMP)/usr/share/doc/$(PACKAGE)/examples -type f -print0 | \
		xargs -r0 sed -i -e '1s|^#!/usr/local/bin/perl|#!/usr/bin/perl|'
```

Parallel building

Some upstream build systems are not parallel-build safe. debhelper's `--max-parallel` can be used to disable parallel building:

```
override_dh_auto_build:
        dh_auto_build --max-parallel=1
```

Or just `--no-parallel`, available since `9.20151004`:

```
override_dh_auto_build:
        dh_auto_build --no-parallel
```

Note that debhelper does **not** enable parallel building automatically before compat level 10.

Skipping tests

Sometimes we want to skip a specific test, i.e. when it needs internet access. An option to do this is:

```
TEST_FILES = $(filter-out t/foo.t t/bar.t,$(shell echo t/*.t))

override_dh_auto_test:
        #for EUMM:
        dh_auto_test -- TEST_FILES="$(TEST_FILES)"
        #for M::B:
        dh_auto_test -- --test_files "$(TEST_FILES)"
```

For skipping the same tests during build as during **autopkgtest** you can set *TEST_FILES* this way:

```
SKIP_TESTS=$(shell cat debian/tests/pkg-perl/smoke-skip)
TEST_FILES=$(filter-out $(SKIP_TESTS), $(wildcard t/*.t))
```

Running tests needing writable HOME

Sometimes tests need to have access to a writable `$HOME`.

```
BUILDHOME = $(CURDIR)/debian/build

%:
        dh $@

override_dh_clean:
        dh_clean
        rm -rf $(BUILDHOME)

override_dh_auto_test:
        mkdir -p $(BUILDHOME)
        HOME=$(BUILDHOME) dh_auto_test
```

PS: If `$HOME` is set during build, it should normally be set for **autopkgtests** as well, e.g. by putting `HOME=$ADTTMP` into *debian/tests/pkg-perl/smoke-env*.

CONTRIBUTORS

The following people have contributed to the maintainership of this file:

- Jonathan Yu <jawnsy@cpan.org>
- Tim Retout <tim@retout.co.uk>
- gregor herrmann <gregoa@debian.org>
- Damyan Ivanov <dmn@debian.org>

LICENSE AND COPYRIGHT

Any decisions that need to be made regarding the licensing, copyright and distribution of this file shall be determined by majority vote between the current members of the Debian Perl Team, with one vote for all members, and an additional vote granted to Debian Developers.