NAME

ebuild − a low level interface to the Portage system

SYNOPSIS

ebuild *file command [command]...*

DESCRIPTION

The ebuild program is a direct interface to the Portage system. It allows for direct action upon an ebuild with specific subcommands or groups of commands to perform in a specific ebuild's context and functions. Accepting an ebuild script and one or more commands as arguments, the ebuild program parses the ebuild script and executes the specified commands. Commands exist to fetch sources, unpack sources, compile sources, install object files into a temporary directory "image", merge the image to the local filesystem, create a bzipped tarball package out of the image, and more.

FILE

This must be a valid ebuild script. For further information read **ebuild**(5).

COMMANDS

By default, portage will execute all the functions in order up to the one actually specified, except for the functions that have already been executed in a previous invocation of ebuild. For example, simply issuing the command **compile** will trigger the functions before it to also be run (such as **setup** and **unpack**), unless they were run in a previous invocation of ebuild. If you want to make sure they are all run, you need to use the command **clean** first. If you wish to only have the specified command run, then you should use the *noauto* option in the **FEATURES** environment variable. See the**mak e.conf**(5) man page for more information.

help Shows a condensed form of this man page along with a lot of package specific information.

setup Runs all package-specific setup actions (by running the *pkg_setup()* function specified in the ebuild file) and exotic system checks.

clean Cleans the temporary build directory that Portage has created for this particular ebuild file. The temporary build directory normally contains the extracted source files as well as a possible "install image" (all the files that will be merged to the local filesystem or stored in a package). The location of the build directory is set by the PORTAGE_TMPDIR variable. For information on what this variable is, run *emerge −−info*, or to override this variable, see **make.conf**(5).

Note: Portage cleans up almost everything after a package has been successfully merged unless FEATURES contains 'noclean'. Adding noclean to FEATURES will cause a lot of files to remain and will consume large amounts of space, very quickly. It is not recommended to leave this on, unless you have use for the sources post−merge. Optionally, one may manually clean these files with *rm −rf /var/tmp/portage*.

fetch Checks to see if all the sources specified in SRC_URI are available in DISTDIR (see **make.conf**(5) for more information) and have a valid checksum. If the sources aren't available, an attempt is made to download them from the locations specified in SRC_URI. If multiple download locations are listed for a particular file, Portage pings each location to see which location is closer. (May not be true presently.) The Gentoo Linux mirrors defined by GENTOO_MIRRORS is always considered first. If for some reason the current or just−downloaded sources' checksums don't match those recorded in files/digest−[package]−[version−rev], a warning is printed and ebuild exits with an error code of 1.

digest This is now equivalent to the *manifest* command.

manifest

Updates the manifest file for the package. This creates checksums for all of the files found in the same directory as the current ebuild as well as the recursive contents of the files subdirectory. It also creates checksums for all of the files listed in SRC_URI for each ebuild. For further information regarding the behavior of this command, see the documentation for the *assume−digests* value

of the **FEATURES** variable in **make.conf**(5). See the **−−force** option if you would like to prevent digests from being assumed.

unpack

Extracts the sources to a subdirectory in the *build directory* (BUILD_PREFIX) by running the *src_unpack()* function in the ebuild file. If no src_unpack() function has been specified, a default src_unpack() function is used that extracts all the files specified in SRC_URI. The sources are normally extracted to ${BUILD_PREFIX}/[package]−[version−rev]/work. This particular directory can be referenced by using the ${WORKDIR} variable.

If you're creating an ebuild, you'll want to make sure that the S (source directory) variable defined at the top of your ebuild script points to the directory that contains your extracted sources. This directory is defined by default to be ${WORKDIR}/${P}, so it is not often required. The src_unpack() function is also responsible for making the appropriate patches to the sources so that they're ready for compilation.

prepare

Prepares the extracted sources by running the *src_prepare()* function specified in the ebuild file. When src_prepare() starts, the current working directory will be set to ${S}. This function is supported beginning with **EAPI 2**.

configure

Configures the extracted sources by running the *src_configure()* function specified in the ebuild file. When src_configure() starts, the current working directory will be set to ${S}. This function is supported beginning with **EAPI 2**.

compile

Compiles the extracted sources by running the *src_compile()* function specified in the ebuild file. When src_compile() starts, the current working directory will be set to ${S}. When src_compile() completes, the sources should be fully compiled.

test Runs package-specific test cases by running the *src_test()* function specified in the ebuild file to verify that everything was built properly.

install Installs the package to the temporary *install directory* by running the *src_install()* function. When completed, the *install directory* (${BUILD_PREFIX}/[package]−[version−rev]/image) will contain all the files that should either be merged to the local filesystem or included in a binary package.

preinst Runs package-specific actions (by running the *pkg_preinst()* function specified in the ebuild file) that need to be done before the package is installed into the live filesystem.

instprep

Performs the additional post-install/pre-merge preparations inside the temporary *install directory*. This is intended to be called **after** building binary package but before executing **preinst**.

postinst

Runs package-specific actions (by running the *pkg_postinst()* function specified in the ebuild file) that need to be done after the package is installed into the live filesystem. Usually helpful messages are shown here.

qmerge

This function installs all the files in the *install directory* to the live filesystem. The process works as follows: first, the *pkg_preinst()* function (if specified) is run. Then, the files are merged into the live filesystem, and the installed files' checksums are recorded in */var/db/pkg/${CATEGORY}/${PN}−${PVR}/CONTENTS*. After all the files have been merged, the *pkg_postinst()* function (if specified) is executed.

merge Normally, to merge an ebuild, you need to *fetch*, *unpack*, *compile*, *install* and *qmerge*. If you're simply interested in merging the ebuild, you can use this command, which will perform all these steps for you, stopping along the way if a particular step doesn't complete successfully.

unmerge
This function first executes the *pkg_prerm()* function (if specified). Then it removes all files from the live filesystem that have a valid checksum and mtime in the package contents file. Any empty directories are recursively removed. Finally, it runs *pkg_postrm()* function (if specified). It is safe to merge a new version of a package first and then unmerge the old one. In fact, this is the recommended package upgrade method.

prerm Runs package-specific actions (by running the *pkg_prerm()* function specified in the ebuild file) that need to be executed before the package is removed from the filesystem. See also *unmerge*.

postrm Runs package-specific actions (by running the *pkg_postrm()* function specified in the ebuild file) that need to be executed after the package is removed from the filesystem. See also *unmerge*.

config Runs package-specific actions (by running the *pkg_config()* function specified in the ebuild file) that need to be executed after the emerge process has completed. This usually entails setup of configuration files or other similar setups that the user may wish to run.

package
This command is a lot like the *merge* command, except that after fetching, unpacking, compiling and installing, a .gpkg.tar or .tbz2 binary package tarball is created and stored in **PKGDIR** (see **make.conf**(5)).

rpm Builds a RedHat RPM package from the files in the temporary *install directory*. At the moment, the ebuild's dependency information is not incorporated into the RPM.

OPTIONS

−−debug
Run bash with the −x option, causing it to output verbose debugging information to stdout.

−−color < y | n >
Enable or disable color output. This option will override *NO_COLOR* and *NOCOLOR* (see **make.conf**(5)) and may also be used to force color output when stdout is not a tty (by default, color is disabled unless stdout is a tty).

−−force
When used together with the digest or manifest command, this option forces regeneration of digests for all distfiles associated with the current ebuild. Any distfiles that do not already exist in ${DISTDIR} will be automatically fetched.

−−ignore−default−opts
Do not use the *EBUILD_DEFAULT_OPTS* environment variable.

−−skip−manifest
Skip all manifest checks.

REPORTING BUGS

Please report bugs via https://bugs.gentoo.org/

AUTHORS

Achim Gottinger <achim@gentoo.org>
Daniel Robbins <drobbins@gentoo.org>
Nicholas Jones <carpaski@gentoo.org>
Mike Frysinger <vapier@gentoo.org>

FILES

/etc/portage/make.conf
Contains variables for the build−process and overwrites those in make.globals.

/etc/portage/color.map
Contains variables customizing colors.

SEE ALSO

emerge(1), **ebuild**(5), **make.conf**(5), **color.map**(5)

The */usr/lib/portage/bin/ebuild.sh* script.

The helper apps in */usr/lib/portage/bin*.