

Cleaning Targets

Table 10-2 shows targets that simply remove files from previous builds. Their use is highly recommended to make sure you don't contaminate new builds with files leftover that may have been built with different options. They differ in how much they remove; sometimes you want to keep around files you've changed.

Table 10-2. Cleaning targets

| Target | Description |
|------------------------|---|
| <code>clean</code> | Removes most of the files generated by the kernel build system, but keeps the kernel configuration. |
| <code>mrproper</code> | Removes all of the generated files by the kernel build system, including the configuration and some various backup files. |
| <code>distclean</code> | Does everything <code>mrproper</code> does and removes some editor backup and patch leftover files. |

Configuration Targets

Table 10-3 shows targets that allow the kernel to be configured in a wide range of different ways.

Table 10-3. Configuration targets

| Target | Description |
|------------------------------|--|
| <code>config</code> | Updates the current kernel configuration by using a line-oriented program. |
| <code>menuconfig</code> | Updates the current kernel configuration by using a text-based menu program. |
| <code>xconfig</code> | Updates the current kernel configuration by using a QT-based graphical program. |
| <code>gconfig</code> | Updates the current kernel configuration by using a GTK+-based graphical program. |
| <code>oldconfig</code> | Updates the current kernel configuration by using the current <code>.config</code> file and prompting for any new options that have been added to the kernel. |
| <code>silentoldconfig</code> | Just like <code>oldconfig</code> , but prints nothing to the screen except when a question needs to be answered. |
| <code>randconfig</code> | Generates a new kernel configuration with random answers to all of the different options. |
| <code>defconfig</code> | Generates a new kernel configuration with the default answer being used for all options. The default values are taken from a file located in the <code>arch/\$ARCH/defconfig</code> file, where <code>\$ARCH</code> refers to the specific architecture for which the kernel is being built. |
| <code>allmodconfig</code> | Generates a new kernel configuration in which modules are enabled whenever possible. |
| <code>allyesconfig</code> | Generates a new kernel configuration with all options set to yes. |
| <code>allnoconfig</code> | Generates a new kernel configuration with all options set to no. |

Note that the `allyesconfig`, `allmodconfig`, `allnoconfig`, and `randconfig` targets also take advantage of the environment variable `KCONFIG_ALLCONFIG`. If that variable points to a file, that file will be used as a list of configuration values that you require to be set to a specific value. In other words, the file overrides the normal behavior of the *make* targets.

For example, if the file `~/linux/must_be_set` contains the following variables:

Table 10-4. Build targets (continued)

| Target | Description |
|--------|---|
| TAGS | Builds all of the needed tags that most common text editors can use while editing the source code. |
| cscope | Builds a <i>cscope</i> image, useful in source tree searches, of the source tree for the architecture specified by the configuration file (not all of the kernel source files). |

You can also pass a number of environment variables to *make* that will change the build. These can be specified for almost any target, as shown in Table 10-5.

Table 10-5. Environment variables

| Variable | Value | Description |
|----------|-------|---|
| V | 0 | This tells the build system to run in a quiet manner, showing only the file that is currently being built, and not the entire command that is running in order to build that file. This is the default option for the build system. |
| V | 1 | This tells the build system to operate in a verbose way, showing the full command that is being used to generate each of the specific files. |
| O | dir | This tells the build system to locate all output files in the <i>dir</i> directory, including the kernel configuration files. This allows the kernel to be built from a read-only filesystem and have the output placed in another location. |
| C | 1 | This checks all C files that are about to be built with the <i>sparse</i> tool, which detects common programming errors in the kernel source files. <i>sparse</i> can be downloaded using <i>git</i> from git://git.kernel.org/pub/scm/devel/sparse/sparse.git . Nightly snapshots can be found at http://www.codemonkey.org.uk/projects/git-snapshots/sparse/ . More information on how to use <i>sparse</i> can be found in the <i>Documentation/sparse.txt</i> file in the kernel source tree. |
| C | 2 | This forces all C files to be checked with the <i>sparse</i> tool, even if they did not need to be built. |

Packaging Targets

These targets package up a built kernel into a standalone package that can be installed on a wide range of different machines, as shown in Table 10-6.

Table 10-6. Packaging targets

| Target | Description |
|-------------|--|
| rpm | Builds the kernel first and then packages it up as a RPM package that can be installed. |
| rpm-pkg | Builds a source RPM package containing the base kernel. |
| binrpm-pkg | Builds a RPM package that contains a compiled kernel and modules. |
| deb-pkg | Builds a Debian package that contains the compiled kernel and modules. |
| tar-pkg | Builds a tarball that contains the compiled kernel and modules. |
| tar.gz-pkg | Builds a <i>gzip</i> -compressed tarball that contains the compiled kernel and modules. |
| tar.bz2-pkg | Builds a <i>bzip2</i> -compressed tarball that contains the compiled kernel and modules. |

