Tree Dired

The GNU Emacs Directory Editor

Sebastian Kremer
sk@thp.uni-koeln.de

$Date: 1993/01/30 06:50:14 $
$Revision: 1.2 $

1 Dired, the Directory Editor

Dired makes it easy to delete or visit many of the files in a single directory (and possibly its subdirectories) at once. It makes an Emacs buffer containing a listing of the directories, in the format of `ls -lR`. You can use the normal Emacs commands to move around in this buffer, and special Dired commands to operate on the files. You can run shell commands on files, visit, compress, load or byte-compile them, change their file attributes and insert subdirectories into the same buffer. You can "mark" files for later commands or "flag" them for deletion, either file by file or all files matching certain criteria.

1.1 Entering Dired

To invoke Dired, do *`C-x d`* or *`M-x dired`*. The command reads a directory name or wildcard file name pattern as a minibuffer argument just like the `list-directory` command, *`C-x C-d`*. Invoking Dired with a prefix argument lets you enter the listing switches for the directory.

Dired assumes you meant to use a wildcard if the last component of the name is not an existing file. Note that only the last pathname component may contain wildcards. With wildcards it uses the shell to do the filename globbing, whereas usually it calls '`ls`' directly. Because of this, you might have to quote characters that are special to the shell. For example, to dired all auto-save files in your `~/mail/` directory, use '`~/mail/\#*`' as argument to Dired. Note the backslash needed to quote '`#`' (at the beginning of a word) to the shell.

Where `dired` differs from `list-directory` is in naming the buffer after the directory name or the wildcard pattern used for the listing, and putting the buffer into Dired mode so that the special commands of Dired are available in it. The variable `dired-listing-switches` is a string used as an argument to `ls` in making the directory; this string *must* contain '`-l`'. Most other switches are also allowed, especially '`-F`', '`-i`' and '`-s`'. For the '`-F`' switch to work you may have to set the variable `dired-ls-F-marks-symlinks`, depending on what kind of '`ls`' program you are using. See Section 1.10.2 [Dired Configuration], page 17.

When a Dired buffer for the given directory already exists, it is simply selected without refreshing it. You can type *`g`* if you suspect it is out of date.

To display the Dired buffer in another window rather than in the selected window, use *`C-x 4 d`* (`dired-other-window`) instead of *`C-x d`*.

1.2 Editing in Dired

Once the Dired buffer exists, you can switch freely between it and other Emacs buffers. Whenever the Dired buffer is selected, certain special commands are provided that operate on files that are listed. The Dired buffer is "read-only", and inserting text in it is not useful, so ordinary printing characters such as *`d`* and *`x`* are used for Dired commands, and digits are prefix arguments.

The file described by the line that point is on is called the *current file*. The directory this file is in is the *current Dired directory*. Note that there may be several directories in one Dired buffer as long as they belong to the same tree. The top level directory, the *root* of the tree, is used as the working directory of the buffer.

Some or all directories can be *hidden*, leaving only their headerlines visible, and exlcuding their files from Dired operations.

Files can be *marked* for later commands. Marking means putting a special character, usually '`*`', in the first column of the file line. To *flag* a file means to mark it for later deletion. This special case of "marking" is distinguished so that you do not delete files accidentally. Internally, the only difference between marking and flagging is the character used to mark the file: '`*`' (an asterisk) for a marked file, and '`D`' for files flagged for deletion.

Most Dired commands operate on the "marked" files and default to the current file. They are the *mark-using* commands. Deleting is the only mark-using command that does not default to the current file.

Dired buffers "know" about each other. For example, copying from *dir1* into *dir2* will update *dir2*'s Dired buffer(s). When you move files or directories, file and dired buffers are kept up to date and refer to the new location. But Dired only knows about files changed by itself, not by other parts of Emacs or programs outside Emacs.

All the usual Emacs cursor motion commands are available in Dired buffers. Some special purpose commands are also provided. The keys *C-n* and *C-p* are redefined so that they try to position the cursor at the beginning of the filename on the line, rather than at the beginning of the line.

For extra convenience, `SPC` and *n* in Dired are equivalent to *C-n*. *p* is equivalent to *C-p*. Moving by lines is also done so often in Dired that it deserves to be easy to type. `DEL` (move up and unflag) is often useful simply for moving up.

1.3 Listing Files in Dired

Initially the Dired buffer shows the directory you selected. The first line shows the full directory name. It is an example of a *headerline* of a directory. Note that it is terminated by a colon ('`:`') that is not part of the directory name. The second line usually displays the total size of all files in the directory or the wildcard used. Both are examples of *non-file lines*. Applying a command to a non-file line signals an error. The other lines of the directory, called the *file lines*, show information about each file such as permission bits, size and date of last modification, and the name of the file.

For example, the listing

```
/home/sk/lib/emacs/lisp:
total 4973
-rw-r--r--   1 sk       users      231608 Feb  6 16:58 ChangeLog
drwxr-sr-x   2 sk       users        2048 Feb  6 11:07 RCS
-r--r--r--   1 sk       users      141389 Feb  6 10:45 dired.el
-r--r--r--   1 sk       users      113033 Feb  5 16:21 dired.texi
...

/home/sk/lib/emacs/lisp/RCS:
total 4798
-r--r--r--   1 sk       users      231748 Feb  6 16:59 dired.texi,v
-r--r--r--   1 sk       users      763898 Feb  6 10:45 dired.el,v
...
```

has a headerline for the `lisp` directory, a total line saying there are 4973 K in all the files of that directory (your '`ls`' program may use units of blocks instead), and several file lines. After that, a headerline for the `RCS` subdirectory with its total line and its files follows.

Here is an example of a wildcard listing:

```
/home/sk/lib/emacs/lisp:
wildcard dired*
-rw-r--r--   1 sk       users      113036 Feb  6 16:59 dired.texi
-r--r--r--   1 sk       users       81267 Feb  6 16:29 dired.elc
-r--r--r--   1 sk       users       38436 Feb  6 16:28 dired-x.elc
-r--r--r--   1 sk       users       60258 Feb  6 16:27 dired-x.el
-r--r--r--   1 sk       users      141389 Feb  6 10:45 dired.el
...
```

Since '`ls`' does not provide a total count when called with a wildcard argument, the second line now gives instead the wildcard used, here '`dired*`'. If there would have been a directory matching the wildcard, e.g. a '`dired/`' subdirectory, its file line would be shown, but it would not have been expanded automatically.

Filenames may have embedded and trailing (but not leading) spaces. Leading spaces are not recognized because different '`ls`' programs differ in the amount of whitespace the insert before the filename. Filenames may *not* contain newlines or '`^M`''s. You can get away with '`^M`''s in filenames if you do

```
(setq selective-display nil)
```

in the Dired buffer (inside `dired-mode-hook`, See Section 1.10.3 [Dired Hooks], page 17.). But this also disables the `=` and *$* hiding commands, See Section 1.8 [Hiding Directories in Dired], page 14.

Other unprintable characters than '`^M`' or newline ('`^J`') in filenames are no problem for Dired. But your '`ls`' program may not output them correctly (e.g., replacing all unprintable characters with a question mark '`?`'). Dired can do nothing if '`ls`' suppresses information about the filenames. But some (System V derived) '`ls`' programs have a '`-b`' switch to quote control characters, e.g. '`\n`' for a newline character, or '`\007`' for a ASCII bell character (*C-g*), so you might want to add '`b`' to your switches (see below). Dired translates the quoted control character escapes when a '`-b`' switch was used. The '`-b`' switch is the recommended method to cope with funny filenames containing newlines or leading spaces. But check if your '`ls`' understands '`-b`' and really quotes newlines and spaces. Dired is known to work with GNU '`ls -b`', but other '`ls -b`' don't quote spaces, so leading spaces still don't work with these '`ls`' programs.

The appearance of the listing is determined by the listing switches used, for example whether you display or suppress '`.`' files with the '`-a`' and '`-A`' switches, use the '`-F`' switch to tag filenames etc. It may additionally be restricted to certain files if you used wildcards to display only those files matching a shell file wildcard.

Dired has commands that change the listing switches for this buffer. They are mainly used to set the sort mode, but can also be used to change other formatting options. The buffer is automatically refreshed after the switches are changed to let the new format take effect.

The default value for the switches comes from the variable `dired-listing-switches`; a prefix argument to `dired` can be use to determine the switches used for a specific buffer. See

Section 1.1 [Entering Dired], page 1. Each Dired buffer has its own value for the switches, stored in the variable `dired-actual-switches`.

The Dired modeline displays '`by name`' or '`by date`' to indicate the sort mode. It uses the regexps in the variables `dired-sort-by-date-regexp` and `dired-sort-by-name-regexp` to decide what should be displayed. If neither of the regexps matches, the listing switches are displayed literally. You can use this to always display the literal switches instead of '`by name`' or '`by date`': set them to a regexp that never matches any listing switches, for example '`^$`'.

Most '`ls`' programs can only sort by name (without '`-t`') or by date (with '`-t`'), nothing else. GNU '`ls`' additionally sorts on size with '`-S`', on extension with '`-X`', and unsorted (in directory order) with '`-U`'. So anything that does not contain these is sort "by name". However, this is configurable in the variable `dired-ls-sorting-switches`, which defaults to `"SXU"`. It contains a string of '`ls`' switches (single letters) except '`t`' that influence sorting. It is consulted at load time, so if you redefine it, you must do it before Dired is loaded.

s (`dired-sort-toggle-or-edit`) Toggle between sort by name/date and refresh the dired buffer. With a prefix argument you can edit the current listing switches instead.

After some time the listing may become out of date because of actions by other programs than Dired. You can refresh the complete Dired buffer from disk or only refresh the lines of certain files or a single file.

l (`dired-do-redisplay`) Redisplay all marked (or, with a prefix argument, the next *N*) files. As always, if no files are marked, the current file is used.

If on a headerline, redisplay that subdirectory. In that case, a prefix arg lets you edit the '`ls`' switches used for the new listing.

g (`revert-buffer`) The *g* command in Dired ultimately runs `dired-revert` to reinitialize the buffer from the actual disk directory (or directories). All marks and flags in the Dired buffer are restored, except of course for files that have vanished. Hidden subdirectories are hidden again. See Section 1.8 [Hiding Directories in Dired], page 14.

k (`dired-kill-line-or-subdir`) Kill this line (but not this file). Optional prefix argument is a repeat factor. If file is displayed as expanded subdirectory, kill that as well.

If on a subdirectory line, kill that subdirectory. Reinsert it with *i* (`dired-maybe-insert-subdir`), See Section 1.7 [Subdirectories in Dired], page 13.

Killing a file line means that the line is removed from the Dired buffer. The file is not touched, and the line will reappear when the buffer is refreshed (using *g*, `revert-buffer`). A killed subdirectory will not reappear after reverting the buffer, since *g* only list those subdirectories that were listed before.

M-k (`dired-do-kill`) Kill all marked lines (not files). With a prefix argument, kill all lines not marked or flagged.

(For file marking, See Section 1.4 [Marking Files in Dired], page 5.)

C-x u

C-_ (`dired-undo`) Undo in a Dired buffer. This doesn't recover lost files, it is just normal undo with a temporarily writable buffer. You can use it to recover marks, killed lines or subdirs. In the latter case, you have to do *M-x dired-build-subdir-alist* to parse the buffer again for the new subdirectory list.

1.4 Marking Files in Dired

This section describes commands to mark and unmark single files, and commands to mark several files at once if they match certain criteria. There also is a command to move to the next marked file.

As always, hidden subdirs are not affected. See Section 1.8 [Hiding Directories in Dired], page 14.

m (`dired-mark-subdir-or-file`) If on a file line, mark the current file. A numeric argument tells how many next or previous files to mark. If on a subdirectory header line, mark all its files except '.' and '..'.

u (`dired-unmark-subdir-or-file`) Like *m*, only unmarking instead of marking.

DEL (`dired-backup-unflag`) Move up lines and remove flags there. Optional prefix argument says how many lines to unflag; default is one line.

M-DEL (`dired-unflag-all-files`) Remove a specific or all flags from every file. With an argument, queries for each marked file. Type your help character, usually *C-h*, at that time for help.

*** (`dired-mark-executables`) Mark all executable files. With prefix argument, unflag all those files.

@ (`dired-mark-symlinks`) Mark all symbolic links. With prefix argument, unflag all those files.

/ (`dired-mark-directories`) Mark all directory files except '.' and '..'. With prefix argument, unflag all those files.

%m (`dired-mark-files-regexp`) Mark all files matching *regexp* for use in later commands. A prefix argument means to unmark them instead. `.` and `..` are never marked.

The match is against the non-directory part of the filename. Use '`^`' and '`$`' to anchor matches. Exclude subdirs by hiding them.

This is an Emacs regexp, not a shell wildcard. E.g., use '`\.o$`' for object files - just '`.o`' will mark more than you might think. By default, the match is case sensitive (just like filenames), since `case-fold-search` is set to `nil` in Dired buffers.

M-} (`dired-next-marked-file`) Move to the next marked file, wrapping around the end of the buffer.

M-{ (`dired-prev-marked-file`) Move to the previous marked file, wrapping around the beginning of the buffer.

1.5 Mark-using Commands

Most Dired commands operate on the "marked" files and default to the current file. They are the "mark-using" commands. Deleting is the only mark-using command that does not default to the current file.

Mark-using Dired commands treat a numeric argument as a repeat count, meaning to act on the files of the next few lines instead of on the marked files. That is, when you give a prefix argument the marks are not consulted at all. A negative argument means to operate on the files of the preceding lines. Either set of files is called *marked files* below, whether they really come from marks or from a prefix argument. The prompt of a mark-using command always makes clear which set of files is operated upon: it mentions either the marker character '`*`' or the '`next` *N*' files, where a negative *N* really means the previous -*N* files.

Thus you can use a prefix argument of `1` to apply a command to just the current file, e.g, if you don't want to disturb the other files you marked. As digits are prefix arguments in Dired, simply type *1* followed by the command.

Many mark-using commands treat a prefix of *N=0* specially, since it would otherwise be a no-op.

All mark-using commands display a list of files for which they failed. Type *W* to see why a (mark-using or other) command failed. Error messages from shell commands ('`stderr`') cannot be redirected separately and goes together with the usual output ('`stdout`').

1.5.1 Copy, Move etc. Into a Directory

This section explains commands that create a new file for each marked file, for example by copying (*c*) or moving (*r*) files. They prompt in the minibuffer for a *target* argument, usually the target directory in which the new files are created. But if there is but one marked file, the target may also be a plain file. (Otherwise you could not simply rename or copy a single file within the same directory.) Even with one marked file the target may still be an (existing) directory.

The target prompt displays a *default target* that will be used if you just type *RET*. Normally the default target is the current Dired directory, so if you want to copy into some specific subdirectory, move point into that subdirectory before typing *c*. But if there is a Dired buffer in the next window, and `dired-dwim-target` is true, its current Dired directory is used. This makes it easy to copy from one Dired buffer into another if both are displayed. On the other hand you have to use *C-x 1* to make other Dired buffers vanish if you do not want to have them as default targets. To make Dired never look at the next window, set the variable `dired-dwim-target` to nil ('`dwim`' means Do What I Mean). See Section 1.10.1 [Dired User Options], page 15, on how to set cutomization variables.

As a general rule, Dired will not let you remove or overwrite a file without explicit confirmation. Dired asks you for each existing target file whether or not to overwrite just this file (answer *y* or *n*) or all remaining files (answer *!*). You can also type your help character, usually *C-h*, at that time for help.

c (`dired-do-copy`) Copy the marked (or next *N*) files into a directory, or copy a single file.

Thus, a zero prefix argument (*N-0*) copies nothing. But it toggles the variable `dired-copy-preserve-time`.

See Section 1.10.1 [Dired User Options], page 15, on how to set customization variables.

r (`dired-do-move`) Move the marked files into a directory. If there is just one marked file, rename that file. As the marked files default to the current file, this can also be used to simply rename the current file.

Dired silently changes the visited file name of buffers associated with moved files so that they refer to the new location of the file.

When a directory is renamed, its headerlines in Dired buffers are updated, and any buffers visiting affected files have their visited file name changed to refer to the new location. Their buffer name is changed if no buffer with such a name already exists. Affected files are all those which contain the directory somewhere in their absolute path name.

A zero prefix arguments does not move any files, but toggles the variable `dired-dwim-target`.

H (`dired-do-hardlink`) Make hard links from the target directory to each marked file.

Y (`dired-do-symlink`) Make symbolic links from the target directory to each marked file.

Linking is very similar to copying in that new files are created while the old files stay. If you want each newly copied or linked file to be marked with the same marker that its original has, set the variables `dired-keep-marker-copy`, `dired-keep-marker-hardlink` or `dired-keep-marker-symlink` to `t`. Set them to `nil` to not give these newly created files marks. The default is to mark them with ‘`C`’, ‘`H`’ and ‘`Y`’, respectively.

Moving differs from copying and linking in that the old file is removed as part of the creation of the new file. Thus it makes sense to set the variable `dired-keep-marker-move` to `t` (the default) so that moved files “take their markers with them”.

1.5.2 Renaming (and More) With Regexps

A second class of Commands uses regular expressions to construct a new filename from each marked file. See Section “Regular Expressions” in *The GNU Emacs Manual*. The commands to make new names by regexp conversion are the same as those to make them in another directory, except that they share a prefix char, `%`.

%r (`dired-rename-regexp`) Rename files with regexps

%c (`dired-do-copy-regexp`) Copy files with regexps.

%H (`dired-do-hardlink-regexp`) Make hard links with regexps.

%Y (`dired-do-symlink-regexp`) Make symbolic links with regexps.

These commands prompt in the minibuffer for a *regexp* and a *newname*. For each marked file matching *regexp*, a new filename is constructed from *newname*. The match can be anywhere in the file name, it need not span the whole filename. Use ‘`^`’ and ‘`$`’ to anchor matches that should span the whole filename. Only the first match in the filename is replaced with *newtext*. (It would be easy to change this to replace all matches, but probably harder to use.)

'`\&`' in *newname* stands for the entire text being replaced. '`\d`' in *newname*, where *d* is a digit, stands for whatever matched the *d*'th parenthesized grouping in *regexp*. As each match is found, the user must type a character saying whether or not to apply the command to just this file (*y* or *n*) or to all remaining files(*!*). For help type your help character, usually *C-h*, at that time.

For example, if you want to rename all `.lsp` files to `.el` files, type first *%m* with '`\.lsp$`' as regexp to mark all `.lsp` files. Then type *%r* with '`\.lsp$`' and '`.el`' as *regexp* and *newtext* arguments. Dired will prompt you for each file to be renamed.

Or to append `.old` to all marked files, use *%r* '`$`' *RET* '`.old`' *RET*, replacing the empty string at the end of each file name with '`.old`'.

You can use the regexp '`\(.+\)\.\(.+\)$`' to make the basename as '`\1`' and the extension as '`\2`' available in *newtext*.

With a zero prefix arg, renaming by regexp affects the complete pathname. Usually only the non-directory part of file names is used and changed, and renaming only takes place within the current directory. The zero prefix argument can be used to change the directory part as well.

Often you will want to apply the command to all files matching the same *regexp* that you use in the command. Simply use the *%m* command with *regexp* as argument, which will then also be the default for the next regexp using command.For example, to remove a `V17I12-` prefix from several filenames, use *%m* '`^V17I12-`' *RET* *%r* *RET* *RET*, in effect replacing the prefix with the empy string.

1.5.3 Other File Creating Commands

Commands to change the case of file names:

%u (`dired-upcase`) Rename each marked file to upper case.

%l (`dired-downcase`) Rename each marked file to lower case.

1.5.4 Deleting Files With Dired

Deleting is a special mark-using command. It uses a special marker, '`D`', and does not default to the current file if no files are marked to prevent accidental deletions.

See Section 1.10 [Dired Customization], page 14, variable `dired-del-marker` to make deleting behave exactly like any mark-using command.

d (`dired-flag-file-deleted`) Flag this file for deletion. If on a subdirectory headerline, mark all its files except `.` and `..`.

u (`dired-unmark-subdir-or-file`) Remove deletion-flag on this line.

DEL (`dired-backup-unflag`) Remove deletion-flag on previous line, moving point to that line.

%d (`dired-flag-regexp-files`) Flag all files containing the specified *regexp* for deletion.

The match is against the non-directory part of the filename. Use '`^`' and '`$`' to anchor matches. Exclude subdirs by hiding them.

The special directories `.` and `..` are never flagged.

x (`dired-do-deletions`) Delete the files that are flagged for deletion (with '`D`').

X (`dired-do-delete`) Delete the '`*`'-marked (as opposed to the '`D`'-flagged) files.

`#` (`dired-flag-auto-save-files`) Flag all auto-save files (files whose names start and end with '`#`') for deletion (see Section "Auto Save" in *The GNU Emacs Manual*).

`~` (`dired-flag-backup-files`) Flag all backup files (files whose names end with '`~`') for deletion (see Section "Backup" in *The GNU Emacs Manual*).

`.` (Period) (`dired-clean-directory`) Flag excess numeric backup files for deletion. The oldest and newest few backup files of any one file are exempt; the middle ones are flagged.

You can flag a file for deletion by moving to the line describing the file and typing *d* or *C-d*. The deletion flag is visible as a '`D`' at the beginning of the line. Point is moved to the beginning of the next line, so that repeated *d* commands flag successive files.

The files are flagged for deletion rather than deleted immediately to avoid the danger of deleting a file accidentally. Until you direct Dired to delete the flagged files, you can remove deletion flags using the commands *u* and `DEL`. *u* works just like *d*, but removes flags rather than making flags. `DEL` moves upward, removing flags; it is like *u* with numeric argument automatically negated.

To delete the flagged files, type *x*. This command first displays a list of all the file names flagged for deletion, and requests confirmation with *yes*. Once you confirm, all the flagged files are deleted, and their lines are deleted from the text of the Dired buffer. The shortened Dired buffer remains selected. If you answer *no* or quit with *C-g*, you return immediately to Dired, with the deletion flags still present and no files actually deleted.

Deletions proceed from the end of the buffer, so if subdirs are in a natural order in the buffer, it usually works to flag '`dir1`', '`dir1/dir2`' and '`dir1/dir2/*`' (by typing *d* on the directory headerlines) and delete everything, including '`dir1/dir2`' and '`dir1`'. Using shell commands (e.g. '`rm -rf`') to remove complete directories may be quicker than having Dired remove each file separately. (See Section 1.5.5 [Dired Shell Commands], page 10.) However, like all actions external to Dired, this does not update the display.

The `#`, `~` and `.` commands flag many files for deletion, based on their names. These commands are useful precisely because they do not actually delete any files; you can remove the deletion flags from any flagged files that you really wish to keep.

`#` flags for deletion all files that appear to have been made by auto-saving (that is, files whose names begin and end with '`#`'). `~` flags for deletion all files that appear to have been made as backups for files that were edited (that is, files whose names end with '`~`').

`.` (Period) flags just some of the backup files for deletion: only numeric backups that are not among the oldest few nor the newest few backups of any one file. Normally `dired-kept-versions` (not `kept-new-versions`; that applies only when saving) specifies the number of newest versions of each file to keep, and `kept-old-versions` specifies the number of oldest versions to keep. Period with a positive numeric argument, as in *C-u 3 .*, specifies the number of newest versions to keep, overriding `dired-kept-versions`. A negative numeric argument overrides `kept-old-versions`, using minus the value of the argument to specify the number of oldest versions of each file to keep.

1.5.5 Shell Commands on Marked files

You can run arbitrary shell commands on the marked files. If there is output, it goes to a separate buffer.

! (`dired-do-shell-command`) Run a shell command on the marked files.

A command string is prompted for in the minibuffer. The list of marked files is appended to the command string unless asterisks '`*`' indicate the place(s) where the list should go. Thus,

```
command -flags
```

is equivalent to

```
command -flags *
```

The filenames are inserted in the order they appear in the buffer. The file listed topmost in the buffer will be the leftmost in the list.

Currently, there is no way to insert a real '`*`' into the command.

As with all mark-using commands, if no files are marked or a specific numeric prefix arg is given, the current or the next *N* files are used. The prompt mentions the file(s) or the marker, as appropriate.

However, for shell commands, a zero argument is special. It means to run command on each marked file separately:

```
cmd * |foo
```

results in

```
cmd F1 |foo; ...; cmd Fn |foo
```

Usually

```
cmd F1 ... Fn |foo
```

would be executed.

No automatic redisplay is attempted because Dired cannot know what files should be redisplayed for a general shell command. For example, a '`tar cvf`' will not change the marked files at all, but rather create a new file, while a '`ci -u -m'...' *`' will probably change the permission bits of all marked files.

Type *l* to redisplay just the marked files, or *l* on a directory headerline to redisplay just that directory, or *g* to redisplay all directories.

The shell command has the top level directory as working directory, so output files usually are created there instead of in a subdirectory, which may sometimes be surprising if all files come from the same subdirectory. Just remember that an Emacs buffer can have but one working directory, and this is the top level directory in Dired buffers.

Examples for shell commands:

- Type *!* and

  ```
  tar cvf foo.tar
  ```

  to tar all marked files into a `foo.tar` file. Dired does not know that a new file has been created and you have to type *g* to refresh the listing. If you have several subdirectories in your Dired buffer, the names given to '`tar`' will be relative to the top level directory, and the output file `foo.tar` will also be created there.

You can use

```
tar cvf - * | compress -c > foo.tar.Z
```

as an alternative to immediately compress the tar file.

- Type *0 !* and

  ```
  uudecode
  ```

  to uudecode each of the marked files. Note the use of the zero prefix argument to apply the shell command to each file separately (uudecode doesn't accept a list of input files). Type *g* afterwards to see the created files.
- Type *0 !* and

  ```
  uuencode * * >*.uu
  ```

  to uuencode each of the marked files, writing into a corresponding `.uu` file. Note the use of the zero prefix argument to apply the shell command to each file separately. Type *g* afterwards to see the created `.uu` files.
- Type *1 !* and

  ```
  mail joe@somewhere <*
  ```

  to mail the current file (note the prefix argument '`1`') to user '`joe@somewhere`'.
- Here is a Dired shell command to execute the current file, assuming no other files are marked (else just give the prefix *1* to *!*):

  ```
  ./*
  ```

  which will be expanded to '`./`*`cmd`*', thus *cmd* will be executed.. (Just '`./`' would be expanded to '`./` *`cmd`*', with an intervening *SPC*.) This will work even if you don't have `.` in your `$PATH`. If `.` is in your path (not a good idea, as you will find out if you dired a directory containing a file named `ls`), a single *SPC* as command would also work.

1.5.6 Compressing and Uncompressing

You can compress or uncompress the marked files. Dired refuses to compress files ending in `.Z` (which are already compressed) or symbolic links (the link would be overwritten by a plain, compressed file) and to uncompress files not ending in `.Z`.

C (`dired-do-compress`) Compress the marked files.

U (`dired-do-uncompress`) Uncompress the marked files.

1.5.7 Changing File Attributes

You can change the file attributes (mode, group, owner) of marked files.

M (`dired-do-chmod`) Change the mode (also called "permission bits") of the marked files. This calls the '`chmod`' program, thus symbolic modes like '`g+w`' are allowed.

Multiple switches like '`-fR g+w`' are not understood, though. Use *!* (`dired-do-shell-command`) for that.

G (`dired-do-chgrp`) Change the group of the marked files.

O (`dired-do-chown`) Change the owner of the marked files. This usually works for the superuser only. It uses the program in the variable `dired-chown-program` to do the change.

1.5.8 Loading and Byte-compiling Emacs Lisp Files

You can load and byte-compile GNU Emacs Lisp files. Errors are caught and reported after all files have been processed.

L (`dired-do-load`) Load the marked elisp files.

B (`dired-do-byte-compile`) Byte compile the marked elisp files.

1.5.9 Printing the Marked Files

P (`dired-do-print`) Print the marked (or next *N*) files. Uses the shell command coming from variables `lpr-command` and `lpr-switches` as default.

Since internally this is just a special case of `dired-do-shell-command`, you can use '`*`' and pipes like for shell command, e.g.,

```
(setq lpr-command: "lwf")
(setq lpr-switches: '("-l -m * | lpr -Palw"))
```

to print with the shell command '`lwf -l -m * | lpr -Palw`', where '`*`' will be substituted by the marked files. The `lpr-buffer` and `lpr-region` don't know about '`*`' or '`|`', though, only Dired does.

1.6 Commands That Do Not Use Marks

These are commands that visit files. See Section "Visiting" in *The GNU Emacs Manual*.

f (`dired-advertised-find-file`) Visits the file described on the current line. It is just like typing *C-x C-f* and supplying that file name. If the file on this line is a subdirectory, *f* actually causes Dired to be invoked on that subdirectory.

o (`dired-find-file-other-window`) Like *f*, but uses another window to display the file's buffer. The Dired buffer remains visible in the first window. This is like using *C-x 4 C-f* to visit the file. See Section "Windows" in *The GNU Emacs Manual*.

v (`dired-view-file`) Views the file described on this line using *M-x view-file*. Viewing a file is like visiting it, but is slanted toward moving around in the file conveniently and does not allow changing the file. See Section "Miscellaneous File Operations" in *The GNU Emacs Manual*. Viewing a file that is a directory goes to its headerline if it is in this buffer. Otherwise, it is displayed in another buffer.

Commands to diff a file:

D (`dired-diff`) Compare file at point with another file (default: file at mark), by running the system command '`diff`'. The other file is the first file given to '`diff`'.

M-~ (`dired-backup-diff`) Diff this file with its backup file. Uses the latest backup, if there are several numerical backups. If this file is a backup, diff it with its original. The backup file is the first file given to '`diff`'.

Other commands:

`+` (`dired-create-directory`) Create a directory.

`W` (`dired-why`) Pop up a buffer with error log output from Dired. All mark-using commands log errors there. (Standard error from shell commands cannot be logged separately, it goes into the usual shell command output buffer.) A group of errors from a single command ends with a formfeed, so that you can use *`C-x [`* (`backward-page`) to find the beginning of new error logs that are reported by a command.

1.7 Subdirectories in Dired

Thise section explains how to *insert* (or *expand*) subdirectories in the same Dired buffer and move around in them.

You can display subdirectories in your Dired buffer by using '`-R`' in your Dired listing switches. But you do not usually want to have a complete recursive listing in all your Dired buffers. So there is a command to insert a single directory:

`i` (`dired-maybe-insert-subdir`) Insert this subdirectory into the same Dired buffer. If it is already present, just move to it (type *`l`*, `dired-do-redisplay` to refresh it). Else inserts it as '`ls -lR`' would have done. With a prefix arg, you may edit the ls switches used for this listing. You can add '`R`' to the switches to expand the whole tree starting at this subdirectory. This function takes some pains to conform to '`ls -lR`' output. For example, it adds the headerline for the inserted subdirectory.

The mark is dropped before moving, so *`C-x C-x`* takes you back to the old position in the buffer.

Dired changes the buffer-local value of the variable `page-delimiter` to `"\n\n"`, so that subdirectories become pages. Thus, the page moving commands *`C-x [`* and *`C-x ]`* (`backward-page` and `forward-page`) can be used to move to the beginning (i.e., the headerlines) of subdirectories.

In addition, the following commands move around directory-wise, usually putting you on a file line instead of on a headerline. For a mnemonic, note that they all look like rotated versions of each other, and that they move in the direction they point to.

`<` (`dired-prev-dirline`) Goto previous directory file line.

`>` (`dired-next-dirline`) Goto next directory file line.

`^` (`dired-up-directory`) Dired parent directory. Tries first to find its file line, then its header line in this buffer, then its Dired buffer, finally creating a new Dired buffer if necessary.

`v` (`dired-view-file`) When the current file is not a directory, view it. When file is a directory, tries to go to its subdirectory.

This command is inverse to the `^` command and it is very convenient to use these two commands together.

The following commands move up and down in the directory tree:

`M-C-u` (`dired-tree-up`) Go up to the parent directory's headerline.

`M-C-d` (`dired-tree-down`) Go down in the tree, to the first subdirectory's headerline.

The following commands move forwards and backwards to subdirectory headerlines:

M-C-n (`dired-next-subdir`) Go to next subdirectory headerline, regardless of level.

M-C-p (`dired-prev-subdir`) Go to previous subdirectory headerline, regardless of level.

1.8 Hiding Directories in Dired

Hiding a subdirectory means to make it invisible, except for its headerline. Files inside a hidden subdirectory are never considered by Dired. For example, mark-using commands will not "see" files in a hidden directory. Thus you can use hiding to temporarily exclude subdirectories from operations without having to remove the markers.

The hiding commands toggle, that is they unhide what was hidden and vice versa.

$ (`dired-hide-subdir`) Hide or unhide the current subdirectory and move to next directory. Optional prefix argument is a repeat factor.

`=` (`dired-hide-all`) Hide all subdirectories, leaving only their header lines. If there is already something hidden, make everything visible again. Use this command to get an overview in very deep directory trees or to move quickly to subdirs far away.

1.9 Acknowledgement

I would like to thank

- Richard Stallman for providing a pre-release version of `dired.el` from Emacs 19, critical comments and many helpful suggestions
- Andy Norman for the collaboration that made Tree Dired and ange-ftp such a successful combination
- Jamie Zawinski for insisting on and writing nested Dired format, and for lots of other things
- Michael Ernst for the "omitting" code and helpful discussion about Dired design
- Hans Chalupsky for providing FTP service and writing `dired-trns.el`
- Roland McGrath for `find-dired.el` and bug fixes for the diffing commands
- Kevin Gallagher for sending me existing VMS Dired fixes
- Hal R. Brand for VMS support and porting his old dired fixes to Tree Dired
- Hugh Secker-Walker for writing `dired-cd.el`
- Tom Wurgler for ideas such as the *dired-jump-back* command
- Cengiz Alaettinoglu, who found more bugs in Tree Dired than anybody else (except me)

and all other beta testers and people who reported bugs or just said "thanks".

1.10 Dired Customization

You can customize Dired by setting some variables in your `~/.emacs` file. Other variables are intended to be configured when Dired is installed. Finally, there are so-called 'hook' variables.

1.10.1 Customization of Dired

The following variables are for personal customization in your `~/.emacs` file. For example, include a line similar to the following

```
(setq dired-listing-switches "-Alt")  ; sort on time, ignore . and ..
```

to set your favorite Dired listing switches.

`dired-listing-switches`

Default: `"-al"`

Switches passed to '`ls`' for Dired. *Must* contain the '`l`' option.

`dired-trivial-filenames`

Default: `"^\\.\\.?$\\|^#"`

Regexp of files to skip when moving point to the first file of a new directory listing. Nil means move to the subdirectory line, t means move to first file.

`dired-marker-char`

Default: `?*` ('`?*`' is the Lisp notation for the character '`*`'.)

In Dired, character used to mark files for later commands.

This is a variable so that one can write things like

```
(let ((dired-marker-char ?X))
   ;; great code using X markers ...
   )
```

`dired-del-marker`

Default: `?D`

Character used to flag files for deletion.

Usually, marking for commands and flagging for deletion are separate features. (Internally they use the same marking mechanism.) You type *d* to flag with '`D`' and *x* to delete the '`D`'-flagged files.

This explains how to make deletion behave just like a special case of the general file marking feature, so that you type *m* to mark with '`*`' (as usual) and *d* to delete the '`*`' (or next *N*) files: In your `~/.emacs`, include

```
(setq dired-del-marker dired-marker-char) ; use * also for deletions
(setq dired-load-hook
      (function
       (lambda ()
         ;; other customizations here
         ;; let "d" do the actual deletion:
         (define-key dired-mode-map "d" 'dired-do-delete)))
```

If you do not like that *d* defaults to the current file if there are no marks, replace the `define-key` statement in `dired-load-hook` above with this one:

```
(define-key dired-mode-map "d" 'dired-do-deletions)
```

`dired-shrink-to-fit`

Default: `(if (fboundp 'baud-rate) (> (baud-rate) search-slow-speed) t)`

Whether Dired shrinks the display buffer to fit the marked files.

`dired-no-confirm`
: Default: `nil`

  If non-nil, list of commands Dired should not confirm. Confirmation for commands that require an argument to be entered (like the shell command for *!*) means a list of marked files is displayed in a pop-up buffer. Confirmation for commands that do not require an argument (like compressing with *C*) means you have to confirm by typing *y* or *SPC*.

  Except `nil`, it can be a sublist of

```
'(byte-compile chgrp chmod chown compress copy delete hardlink load
  move print shell symlink uncompress)
```

  to suppress confirmation for just those commands.

`dired-keep-marker-move`
: Default: `t`

  If nil, moved files are not marked.

  If t, moved marked files are marked with the same marker they had before (maybe none if you used the prefix argument to specify the next *N* files).

  If a character, moved files (marked or not) are marked with that character.

  This also applies to the following, similar variables for copied, and hard or symbolically linked files:

`dired-keep-marker-copy`
: Default: `?C`

`dired-keep-marker-hardlink`
: Default: `?H`

`dired-keep-marker-symlink`
: Default: `?Y`

`dired-dwim-target`
: Default: `nil`

  If non-nil, Dired tries to guess a default target directory: If there is a Dired buffer displayed in the next window, use its current subdirectory, instead of the current subdirectory of this Dired buffer.

  The target is used in the prompt for file copy, move etc., See Section 1.5.1 [Copy and Move Into a Directory], page 6.

`dired-copy-preserve-time`
: Default: `nil`

  If non-nil, Dired preserves the last-modified time in a file copy. (This works on only some systems.) Use *c* (`dired-do-copy`) with a zero prefix argument to toggle its value. The prompt of copy commands will display '`Copy [-p]`' instead of just '`Copy`' if preservation of file times is turned on.

`dired-backup-if-overwrite`
: Default: `nil`

  Non-nil if Dired should ask about making backups before overwriting files. Special value `'always` suppresses confirmation.

1.10.2 Dired Configuration

The following variables should have already been installed correctly by your system manager. If not, you can still set them in your `~/.emacs` file.

`dired-chown-program`
: Pathname of chown command, default `"chown"` (or `"/etc/chown"` on System V derived systems.)

`dired-ls-program`
: Absolute or relative name of the '`ls`' program used by Dired, default `"ls"`.

`dired-ls-F-marks-symlinks`
: Set this to `t` if dired-ls-program with '`-lF`' marks the symbolic link itself with a trailing '`@`' (usually the case under Ultrix).

  Example: If

  ```
  ln -s foo bar; ls -F bar
  ```

  gives

  ```
  bar -> foo
  ```

  set it to `nil`, if it gives

  ```
  bar@ -> foo
  ```

  set it to `t`.

  Dired checks if there is really a @ appended. Thus, if you have a marking '`ls`' program on one host and a non-marking one on another host, and do not care about symbolic links which really end in a @, you can always set this variable to `t`.

1.10.3 Dired Hooks

Hook variables can contain functions that are run at certain times in Dired.

`dired-load-hook`
: Run after loading Dired. You can customize key bindings or load extensions with this. For example:

  ```
  (setq dired-load-hook
        (function
         (lambda ()
          ;; Load extras:
          (load "dired-x")
          ;; How to define your own key bindings:
          (define-key dired-mode-map " " 'scroll-up)
          (define-key dired-mode-map "b" 'scroll-down))))
  ```

`dired-mode-hook`
: Run at the very end of `dired-mode`, after most buffer local variables have been initialized (e.g., `default-directory` and `dired-directory`), but before the directory listing has been read in.

  Do buffer local things here, for example:

  ```
  (setq dired-mode-hook
  ```

```
          (function
           (lambda ()
             (dired-extra-startup)  ;; dired-extra support
             ;; How to set (local) variables in each new Dired buffer:
             (setq case-fold-search t)
             (setq truncate-lines t))))
```

Since the listing has not yet been inserted you could still change `dired-actual-switches`. For example, if you use `ange-ftp.el`, you might want to replace the '`-A`' with the '`-a`' switch, depending on whether `default-directory` corresponds to a System V hosts that does not understand all BSD '`ls`' switches. The `dired.README` file gives an example. If you set `dired-actual-switches` remember that you may also have to set `dired-sort-mode` to the appropriate string so that the modeline looks right.

Do not set `dired-mode-hook` inside your `dired-load-hook`, simply set it somewhere in your `~/.emacs` (before Dired is loaded, if you explicitly load Dired). This is so that extensions packages loaded via the load hook can add things to the `dired-mode-hook` at the front or at the end, as they see fit.

In case you set `truncate-lines` to `t` as in the above example, here is a function to toggle the value of `truncate-lines`, in Dired and other buffers:

```
(defun set-truncate-lines ()
  "Toggle value of truncate-lines and refresh window display."
  (interactive)
  (setq truncate-lines (not truncate-lines))
  ;; now refresh window display (an idiom from simple.el):
  (save-excursion
    (set-window-start (selected-window)
                      (window-start (selected-window)))))
```

You could bind it to *C-x 4 $*:

```
(define-key ctl-x-4-map "$" 'set-truncate-lines)
```

It is sometimes useful to toggle `truncate-lines` in Dired buffers to make long filenames completely visible and get the listing properly aligned again.

`dired-before-readin-hook`
: This hook is run before a dired buffer is newly read in (created or reverted).

`dired-after-readin-hook`
: After each listing of a file or directory, this hook is run with the buffer narrowed to the listing.

The `dired-subdir-alist` has already been updated so that the usual Dired functions like `dired-get-filename` work. It is possible to modify the buffer with this hook. The package `dired-x.el` does this to implement omitting certain uninteresting files from a Dired buffer. Under X11, highlighting of certain files is also possible (see package `dired-x11.el`).

2 Tree Dired Extra features

Numerous "extra" features are available, such as omitting certain files from listings, minibuffer history, RCS related commands, and more.

2.1 Tree Dired Extra Features

The rest of this manual describes the extra features provided by the file `dired-x.el` and some other files.

To take advantage of these features, you must load the file and set some variables and hooks. See the accompanying `dired-x.README` file for details and a template of code to insert in your `.emacs`.

Miscellanous features not fitting anywhere else:

Variables:

`dired-find-subdir`
: Default: `nil`

  If non-nil, Dired does not make a new buffer for a directory if it can be found (perhaps as subdirectory) in some existing Dired buffer.

  If there are several Dired buffers for a directory, the most recently used is chosen.

  Dired avoids switching to the current buffer, so that if you have a normal and a wildcard buffer for the same directory, *`C-x d RET`* will toggle between those two.

`M-g`
: (`dired-goto-file`) Goto file line of a file (or directory).

`M-G`
: (`dired-goto-subdir`) Goto headerline of an inserted directory. This commands reads its argument with completion over the names of the inserted subdirectories.

`&`
: (`dired-do-background-shell-command`) Run a shell command on the marked files, in the background. This requires `background.el` from Olin Shiver's comint package to work. Note that you can type input to the command in its buffer.

`w`
: (`dired-copy-filename-as-kill`) The *`w`* command puts the names of the marked (or next *N*) files into the kill ring, as if you had killed them with *`C-w`*. With a zero prefix argument N=0, use the complete pathname of each file. With a raw (just *`C-u`*) prefix argument, use the relative pathname of each marked file. As a special case, if no prefix argument is given and point is on a directory headerline, it gives you the name of that directory, without looking for marked files.

  The list of names is also stored onto the variable `dired-marked-files` for use, e.g., in an *`ESC ESC`* (`eval-expression`) command.

  As this command also displays what was pushed onto the kill ring you can use it to display the list of currently marked files in the echo area (unless you happen to be on a subdirectory headerline).

You can then feed the file name to other Emacs commands with *C-y*. For example, say you want to rename a long filename to a slightly different name. First type *w* to push the old name onto the kill ring. Then type *r* to rename it and use *C-y* inside *r*'s minibuffer prompt to insert the old name at a convenient place.

T (`dired-do-toggle`) Toggle marks. That is, currently marked files become unmarked and vice versa. Files marked with other flags (such as 'D') are not affected. The special directories '.' and '..' are never toggled.

2.2 Minibuffer History for Dired Shell Commands

If `dired-x.el` determines at load-time that the Gmhist package is available, the Dired shell commands *!* and *&* maintain a history of commands. Use *M-p* and *M-n* to scroll through them while you are prompted in the minibuffer for a shell command. See Section "Gmhist Keys in the Minibuffer" in *The Gmhist Manual*, for more info.

Gmhist also handles defaults:

`dired-dangerous-shell-command`
Default: `"rm"`

Regexp for dangerous shell commands that should never be the default. It is deliberately chosen to match '`rm`' anywhere in the command, e.g. in '`rmdir`'.

2.3 Insert all marked subdirectories

I (`dired-do-insert-subdir`) Insert all marked subdirectories that are not already inserted. Non-directories are silently ignored.

Thus type */I* to insert one more level of subdirectories. You can repeat this until no new directories are inserted to fully expand the directory tree in this buffer. As a faster alternative, use the prefix argument for *C-x d* (`dired`) to add '`R`' to the switches.

2.4 Dynamic Marker Characters

You can change the marker character from its usual value `*` to something else. Use this to mark a different set of files while keeping the information on the already marked files. You can nest several marker characters. The current stack of marker characters is displayed in the Dired mode line, and all prompts of mark-using commands mention to which marker they apply.

((`dired-set-marker-char`) Set the marker character to something else. Use *)* to restore the previous value.

) (`dired-restore-marker-char`) Restore the marker character to its previous value. Uses `dired-default-marker` if the marker stack is empty.

Instead of using *m* inside a *(...)*, you can mark files "in passing" with, say '`Z`' without changing the current marker character. You will probably later use *(* to temporarily make '`Z`' to the marker and do something on the '`Z`'-files, and then return using `)`.

`dired-mark-keys`
Default: `'("Z")`

List of keys (strings) that insert themselves as file markers.

2.5 Omitting Files in Dired

Omitting a file means removing it from the directory listing. Omitting is useful for keeping Dired buffers free of uninteresting files (for instance, auto-save, auxiliary, backup, and revision control files) so that the user can concentrate on the interesting files. Like hidden files, omitted files are never seen by Dired. See Section "Hiding in Dired" in *Tree Dired Manual*. Omitting differs from hiding in several respects:

- Omitting works on individual files, not on directories; an entire directory cannot be omitted (though each of its files could be).
- Omitting is wholesale; if omitting is turned on for a dired buffer, then all "uninteresting" files listed in that buffer are omitted. The user does not omit (or unomit) files one at a time.
- Omitting can be automatic; uninteresting file lines in the buffer can be removed before the user ever sees them.
- Marked files are never omitted.

`M-o`
: (`dired-omit-toggle`) Toggle between displaying and omitting "uninteresting" files. With a prefix argument, don't toggle and just mark the files, but don't actually omit them.

In order to make omitting work, you must have `dired-omit-expunge` on your `dired-after-readin-hook`, and you must call `dired-omit-startup` (or `dired-extra-startup`, which calls `dired-omit-startup`) in your `dired-mode-hook`. Simply loading `dired-x.el` inside `dired-load-hook` takes care of all this.

The following variables can be used to customize omitting.

`dired-omit-files-p`
: Default: `nil`

  If non-nil, "uninteresting" files are not listed. Uninteresting files are those whose filenames match regexp `dired-omit-files`, plus those ending with extensions in `dired-omit-extensions`. `M-o` (`dired-omit-toggle`) toggles its value, which is buffer-local. Do

```
(setq dired-omit-files-p t)
```

  inside your `dired-mode-hook` to have omitting initially turned on in every Dired buffer. Since `dired-x.el` prepends the form '`(dired-extra-startup)`' to what you put yourself in your `dired-mode-hook`, the `setq` will take place after `dired-omit-files-p` has already been made local to the current Dired buffer, so modelines of non-dired buffers are not affected. For this to work you shouldn't set `dired-mode-hook` inside `dired-load-hook`, but directly in your `~/.emacs` (before Dired is loaded, if you explicitly load Dired).

  You can then use `M-o` to unomit in that buffer.

`dired-omit-files`
: Default: `"^#\\|\\.$"`

  Filenames matching this buffer-local regexp will not be displayed. This only has effect when `dired-omit-files-p` is t.

The default value omits the special directories `.` and `..` and autosave files (plus other files ending in “.”).

`dired-omit-extensions`
: Default: The elements of `completion-ignored-extensions`, `latex-unclean-extensions`, `bibtex-unclean-extensions` and `texinfo-unclean-extensions`.

  If non-nil, a list of extensions (strings) to omit from Dired listings. Its format is the same as that of `completion-ignored-extensions`.

`dired-omit-localp`
: Default: `'no-dir`

  The *localp* argument `dired-omit-expunge` passes to `dired-get-filename`. If it is `'no-dir`, omitting is much faster, but you can only match against the non-directory part of the filename. Set it to `nil` if you need to match the whole pathname or `t` to match the pathname relative to the buffer's top-level directory.

`dired-omit-marker-char`
: Default: *`C-o`*

  Temporary marker used by by Dired to implement omitting. Should never be used as marker by the user or other packages. There is one exception to this rule: by doing

```
(setq dired-mark-keys "\C-o")
;; i.e., the value of dired-omit-marker-char
;; (which is not defined yet)
```

  anywhere in your `~/.emacs`, you will bind the *`C-o`* key to insert a `C-o` marker, thus causing these files to be omitted in addition to the usually omitted files. Unfortunately the files you omitted manually this way will show up again after reverting the buffer, unlike the others.

To avoid seeing RCS files and the RCS directory, do

```
(setq dired-omit-files "\\.$\\|#\\|^RCS$\\|,v$")
```

This assumes `dired-omit-localp` has its default value of `'no-dir` to make the `^`-anchored matches work. As a slower alternative, with `dired-omit-localp` set to `nil`, you can use `/` instead of `^` in the regexp.

If you use tib, the bibliography program for use with TEX and LaTEX, you might want to omit the `INDEX` and the `-t.tex` files:

```
(setq dired-omit-files "\\.$\\|#\\|^INDEX$\\|-t\\.tex$")
```

2.6 Advanced Mark Commands

`M-(`
: (`dired-mark-sexp`) Mark files for which *predicate* returns non-nil. With a prefix argument, unflag those files instead.

  The *predicate* is a lisp expression that can refer to the following symbols:

  `inode` [*integer*] the inode of the file (only for ‘`ls -i`’ output)

`s`	[*integer*] the size of the file for '`ls -s`' output (usually in blocks or, with '`-k`', in KBytes)
`mode`	[*string*] file permission bits, e.g., '`"-rw-r--r--"`'
`nlink`	[*integer*] number of links to file
`uid`	[*string*] owner
`gid`	[*string*] group (If the gid is not displayed by '`ls`', this will still be set (to the same as uid))
`size`	[*integer*] file size in bytes
`time`	[*string*] the time that '`ls`' displays, e.g., '`"Feb 12 14:17"`'
`name`	[*string*] the name of the file
`sym`	[*string*] if file is a symbolic link, the linked-to name, else '`""`'

For example, use

```
(equal 0 size)
```

to mark all zero length files.

To find out all not yet compiled Emacs lisp files in a directory, dired all `.el` files in the lisp directory using the wildcard '`*.el`'. Then use *`M-(`* with

```
(not (file-exists-p (concat name "c")))
```

to mark all `.el` files without a corresponding `.elc` file.

`M-M` (`dired-do-unmark`) Unmark marked files by replacing the marker with another character. The new character defaults to a space, effectively unmarking them.

`,` (`dired-mark-rcs-files`) Mark all files that are under RCS control. With prefix argument, unflag all those files. Mentions RCS files for which a working file was not found in this buffer. Type *`W`* (`dired-why`) to see them again.

`C-m C-c` (*`C-m C-c`* is the suggested binding for `dired-mark-files-compilation-buffer`, it is not bound by default.) Mark the files mentioned in the '`*compilation*`' buffer. With an argument, you may specify the other buffer and your own regexp instead of `compilation-error-regexp`. Use '`^.+$`' (the default with a prefix argument) to match complete lines. In conjunction with narrowing the other buffer you can mark an arbitrary list of files, one per line, with this command. If your regexp contains a subexpression, i.e. '`\(...\)`', that subexpression is taken for the file name, else the whole match is used. Thus you can easily strip pre- and suffixes from filenames by using '*`prefix`*`\(.+\)`*`postfix`*' as regexp.

This is especially useful for a list of files obtained from *`M-x grep`* or output from a similar shell command.

`C-m C-d` (*`C-m C-d`* is the suggested binding for `dired-mark-files-from-other-dired-buffer`, it is not bound by default.) Mark those files in this Dired buffer that have the same name as the marked files in the Dired buffer in the other window.

In short, mark the corresponding files from the other Dired buffer.

F
: (`dired-do-find-file`) Visit all marked files at once, and display them simultaneously. If you want to keep the dired buffer displayed, type *C-x 2* first. If you want just the marked files displayed and nothing else, type *C-x 1* first.

  The current window is split across all files. Remaining lines go to the last window.

  The number of files that can be displayed this way is restricted by the height of the current window and the variable `window-min-height`.

`dired-mark-extension`
: Mark all files with a certain extension for use in later commands. A '`.`' is not automatically prepended to the string entered.

  When called from lisp, *extension* may also be a list of extensions and an optional argument *marker-char* specifies the marker used.

`dired-flag-extension`
: Flag all files with a certain extension for deletion. A '`.`' is *not* automatically prepended to the string entered.

`dired-clean-patch`
: Flag dispensable files created by the '`patch`' program for deletion. See variable `patch-unclean-extensions`.

`dired-clean-tex`
: Flag dispensable files created by TEX, LaTEX and '`texinfo`' for deletion. See variables `tex-unclean-extensions`, `texinfo-unclean-extensions`, `latex-unclean-extensions` and `bibtex-unclean-extensions`.

Variables used by the above cleanup commands (and in the default value for variable `dired-omit-extensions`):

`patch-unclean-extensions`
: Default: `'(".rej" ".orig")`

  List of extensions of dispensable files created by the '`patch`' program.

`tex-unclean-extensions`
: Default: `'(".toc" ".log" ".aux")`

  List of extensions of dispensable files created by TEX.

`texinfo-unclean-extensions`
: Default: `'(".cp" ".cps" ".fn" ".fns" ".ky" ".kys" ".pg" ".pgs" ".tp" ".tps" ".vr" ".vrs")`

  List of extensions of dispensable files created by texinfo.

`latex-unclean-extensions`
: Default: `'(".idx" ".lof" ".lot" ".glo")`

  List of extensions of dispensable files created by LaTeX.

`bibtex-unclean-extensions`
: Default: `'(".blg" ".bbl")`

  List of extensions of dispensable files created by BibTeX.

2.7 Virtual Dired

Using *Virtual Dired* means putting a buffer with Dired-like contents in Dired mode. The files described by the buffer contents need not actually exist. This is useful if you want to peruse an '`ls -lR`' output file, for example one you got from an FTP server. You can use all motion commands usually available in Tree Dired. You can also use it to save a Dired buffer in a file and resume it in a later session.

Type *`M-x dired-virtual`* to put the current buffer into virtual Dired mode. You will be prompted for the top level directory of this buffer, with a default value guessed from the buffer contents. To convert the virtual to a real Dired buffer again, type *`g`* (which calls `dired-virtual-revert`) in the virtual Dired buffer and answer '`y`'. You don't have to do this, though: you can relist single subdirectories using *`l`* (`dired-do-redisplay`) on the subdirectory headerline, leaving the buffer in virtual Dired mode all the time.

The function '`dired-virtual-mode`' is specially designed to turn on virtual Dired mode from the `auto-mode-alist`. To automatically edit all `*.dired` files in virtual Dired mode, put this into your `~/.emacs`:

```
(setq auto-mode-alist (cons '("[^/]\\.dired$" . dired-virtual-mode)
                            auto-mode-alist))
```

The regexp is a bit more complicated than usual to exclude ".dired" local variable files.

2.8 Multiple Dired Directories and Non-Dired Commands

An Emacs buffer can have but one working directory, stored in the buffer-local variable `default-directory`. A Dired buffer may have several subdirectories inserted, but still has but one working directory: that of the top level Dired directory in that buffer. For some commands it is appropriate that they use the current Dired directory instead of `default-directory`, e.g., `find-file` and `compile`.

A general mechanism is provided for special handling of the working directory in special major modes:

`default-directory-alist`
: Default: `((dired-mode . (dired-current-directory)))`

  Alist of major modes and their opinion on `default-directory`, as a lisp expression to evaluate. A resulting value of `nil` is ignored in favor of `default-directory`.

`default-directory`
: Function with usage like variable `default-directory`, but knows about the special cases in variable `default-directory-alist`.

The following dired-x commands take special care about the current Dired directory:

`find-this-file`
: Bind this to *`C-x C-f`* as a replacement for `find-file` that will prompt for the filename within the current Dired subdirectory, not the top level directory.

`find-this-file-other-window`
: Bind this to *`C-x 4 C-f`* as a replacement for `find-file-other-window`.

`dired-smart-shell-command`
: Like function `shell-command`, but in the current Tree Dired directory. Bound to *M-!* in Dired buffers.

`dired-smart-background-shell-command`
: Like function `background`, but in the current Tree Dired directory. Bound to *M-&* in Dired buffers.

`dired-jump-back`
: (Suggested binding *C-x j*) Jump back to dired: If in a file, dired the current directory and move to file's line. If in Dired already, pop up a level and goto old directory's line. In case the proper Dired file line cannot be found, refresh the Dired buffer and try again.

`dired-jump-back-other-window`
: (Suggested binding *C-x 4 j*) Like `dired-jump-back`, but to other window.

`dired-vm`
: (*V*) Run VM on this file (assumed to be a UNIX mail folder). Further 'v' commands from within VM in that folder will default to the folder's directory, not the usual `vm-folder-directory`.

  If you give this command a prefix argument, it will visit the folder read-only. This only works in VM 5, not VM 4.

  If the variable `dired-vm-read-only-folders` is t, `dired-vm` will visit all folders read-only. If it is neither `nil` nor `t`, e.g., the symbol `'if-file-read-only`, only files not writable by you are visited read-only. This is the recommended value if you run VM 5.

`dired-rmail`
: Run Rmail on this file (assumed to be mail folder in Rmail/BABYL format).

2.9 Local Variables for Dired Directories

When Dired visits a directory, it looks for a file whose name is the value of variable `dired-local-variables-file` (default: `.dired`). If such a file is found, Dired will temporarily insert it into the Dired buffer and run `hack-local-variables`. See Section "Local Variables in Files" in *The GNU Emacs Manual*. You can set `dired-local-variables-file` to `nil` to suppress this.

For example, put

```
Local Variables:
dired-actual-switches: "-lat"
dired-sort-mode: " by date"
End:
```

into a `.dired` file of a directory to sort by date only in that directory. Note that since `dired-hack-local-variables` is run inside `dired-mode-hook` the modeline has already been set, so you have to update that for yourself by setting `dired-sort-mode` in addition to changing the switches.

2.10 Making Relative Symbolic Links in Dired

In GNU Emacs version 18, the built-in function `make-symbolic-link` always calls `expand-file-name` on its arguments, so relative symlinks (e.g. '`foo -> ../bar/foo`') are impossible to create.

Dired Extra uses `call-process` and '`ln -s`' for a workaround.

`dired-make-symbolic-link`
: Arguments *name1 name2* and optional *ok-if-already-exists*. Create file *name2*, a symbolic link pointing to *name1* (which may be any string whatsoever and is passed untouched to '`ln -s`'). *ok-if-already-exists* means that *name2* will be overwritten if it already exists. If it is an integer, user will be asked about this. On error, signals a file-error.

`dired-make-relative-symlink`
: Three arguments: *file1 file2* and optional *ok-if-already-exists*. Make a symbolic link *file2* (pointing to *file1*). The link is relative (if possible), for example

```
(dired-make-relative-symlink "/vol/tex/bin/foo"
                             "/vol/local/bin/foo")
```

results in a link

```
/vol/local/bin/foo -> ../../tex/bin/foo
```

`dired-do-relsymlink`
: (binding *`S`*) Symbolically link all marked (or next *N*) files into a directory, or make a symbolic link to the current file. This creates relative symbolic links like

```
foo -> ../bar/foo
```

not absolute ones like

```
foo -> /ugly/path/that/may/change/any/day/bar/foo
```

`dired-do-relsymlink-regexp`
: (*`%S`*) Symbolically link all marked files containing *regexp* to *newname*, using relative (not absolute) names. See functions `dired-rename-regexp` and `dired-do-relsymlink` for more info.

2.11 Letting Dired Guess What Shell Command to Apply

Based upon the name of a filename, Dired tries to guess what shell command you might want to apply to it. For example, if you have point on a file named `foo.tar` and you press *`!`*, Dired will guess you want to '`tar xvf`' it and suggest that as the default shell command.

If you are using the `gmhist` package (See Section 2.2 [Dired Minibuffer History], page 20), the default will be mentioned in brackets and you can type *`M-p`* to get the default into the minibuffer so that you can edit it, e.g., changing '`tar xvf`' to '`tar tvf`'. If there are several commands for a given file, e.g., '`xtex`' and '`dvips`' for a `.dvi` file, you can type *`M-p`* several times to see each of the matching commands.

Dired only tries to guess a command for a single file, never for a list of marked files.

`dired-auto-shell-command-alist-default`

Predefined rules for shell commands. Set this to nil to turn guessing off. The elements of `dired-auto-shell-command-alist` (defined by the user) will override these rules.

`dired-auto-shell-command-alist`

If non-nil, an alist of file regexps and their suggested commands overriding the predefined rules in `dired-auto-shell-command-alist-default`.

Each element of the alist looks like

```
(regexp command...)
```

where each *command* can either be a string or a lisp expression that evaluates to a string. If several *COMMAND*s are given, all will temporarily be pushed on the history.

These rules take precedence over the predefined rules in the variable `dired-auto-shell-command-alist-default` (to which they are prepended when `dired-x` is loaded).

You can set this variable in your `~/.emacs`. For example, to add rules for '`.foo`' and '`.bar`' file extensions, write

```
(setq dired-auto-shell-command-alist
      (list
       (list "\\.foo$" "foo-command");; fixed rule
       ;; possibly more rules...
       (list "\\.bar$";; rule with condition test
              '(if condition
                    "bar-command-1"
                  "bar-command-2"))))
```

This will override any predefined rules for the same extensions.

`dired-guess-have-gnutar`

Default: `nil`

If non-nil, name of the GNU tar executable (e.g., '`"tar"`' or '`"gnutar"`'). GNU tar's '`z`' switch is used for compressed tar files. If you don't have GNU tar, set this to nil: a pipe using '`zcat`' is then used.

2.12 Filename Transformers for Dired Shell Commands

File name *transformers* are functions that take a filename (a string) as an argument and transform it into some other string (e.g., a filename without an extension). This package makes transformers available in Dired shell commands.

For example, running the Dired shell command (type *!* or *M-x* `dired-do-shell-command`)

```
echo * [b] [db]
```

would list the full name, the basename, and the absolute basename of each marked file.

Each transformer is associated with a dispatch character. The associations are stored in a keymap for fast and easy lookup. The dispatch character is used to activate the associated transformer function at a particular position in a shell command issued in Dired. The dispatch character must be enclosed in brackets to distinguish it from normal letters.

To take advantage of this package, simply load it after loading Dired, e.g., in your `dired-load-hook`. You can then use transformers like "[b]" for the basename in your Dired shell commands (see below).

You can define your own transformers using the macro `dired-trans-define`.

`dired-trans-define`
: Macro that assigns the transformer function `(lambda (file)` *body*`)` to *char* (a character or string). *body* must return a string: the transformed file.

Several transformers are predefined:

‘`*`’	returns the unmodified filename (equivalent to ‘`[dbe]`’).
‘`n`’	returns the Name component of a filename without directory information
‘`d`’	returns the Directory component of a filename
‘`b`’	returns the Basename of a filename, i.e., the name of the file without directory and extension (see variable `dired-trans-re-ext`) A basename with directory component can be obtained by ‘`[db]`’.
‘`e`’	returns the Extension of a filename (i.e., whatever `dired-trans-re-ext` splits off)
‘`v`’	returns a file without directory and without `,v` suffixes if any.
‘`z`’	returns a file without directory and without `.Z` suffixes if any.

The following variables can be used to customize `dired-trns.el`:

`dired-trans-re-ext`
: Default: `"\\.[^.]*\\(\\.Z\\)?$"`

  The part of a filename matching this regexp will be viewed as extension.

`dired-trans-starters`
: Default: `"[#[]"`

  User definable set of characters to be used to indicate the start of a transformer sequence.

`dired-trans-enders`
: Default: `"[]# ]"`

  User definable set of characters to be used to indicate the end of a transformer sequence.

2.13 Changing the Working Directory for Dired Shell Commands

The package `dired-cd.el` permits the working directory of the Dired shell commands *!* (`dired-do-shell-command`) and *&* (`dired-do-background-shell-command`) to be the files’ subdirectory under certain circumstances. Loading this extension does not change the behavior of Dired until the variables `dired-cd-same-subdir` and/or `dired-cd-on-each` are non-nil.

If `dired-cd-same-subdir` is non-nil and if all the selected files (marked, non-zero numeric argument, etc.) are in the same subdirectory, then `dired-do-shell-command` and `dired-do-background-shell-command` cause the shell to perform a '`cd`' into that directory before the commands are executed. Also, the selected filenames are provided to the command without any directory components.

If `dired-cd-on-each` is non-nil and if the '`on-each`' option is specified (numeric argument of zero), then *!* (`dired-do-shell-command`) and *&* (`dired-mark-background-shell-command`) use a subshell to perform a '`cd`' into the subdirectory of each file before the commands on that file are executed. Also, each filename is provided to the command without any directory components. Note that this behavior occurs regardless of whether the files are all in the same directory or not.

After the above '`cd`' wrapping has occured, the existing `dired-shell-stuff-it` is used to do the actual file-name quoting and substitution into the command. Thus, custom versions of this procedure should work, e.g., the '`dired-trans`' package will transform commands correctly. However, since filenames lack any directory components, features that use the directory components will fail, e.g. the '`[d]`' transform specifier will be empty.

To use this package, load it in your `dired-load-hook`. Do

```
(setq dired-cd-same-subdir t)
```

and perhaps

```
(setq dired-cd-on-each t)
```

in your `~/.emacs`. By default, `dired-cd` doesn't change the behavior of Dired when it is loaded.

If `dired-cd-same-subdir` is non-nil, then the shell commands '`cd`' to the appropriate directory if all the selected files are in that directory; however, on-each behavior (with zero prefix argument) is not changed.

If `dired-cd-on-each` is non-nil, then each instance of the command for an on-each shell command runs in the file's directory regardless of whether the files are all in the same directory.

2.14 Nested Dired format

[NO DOCUMENTATION YET]

This is still buggy, See Appendix B [Dired Known Problems], page 34.

2.15 Feeding Find Output to Dired

The `find-dired` command runs the '`find`' command in a buffer and starts Dired on the inserted file lines, even while '`find`' is still running. For example, with '`-type d`' as argument, you will get a Dired buffer that contains all subdirectories of a given directory, but none of the other files.

Note that '`find`' just gives you file lines, not inserted subdirectories with associated headerlines as repeated use of the *i* (`dired-maybe-insert-subdir`) command would. Also, the names contain slashes if they are in a subdirectory, which never occurs in a normal Dired buffer. Dired understands these names anyway and you can for example type *f* on such lines as usual. However, while '`find`' is still running you shouldn't type *i* to insert subdirectories, since new '`find`' output is always appended at the end. Use *f* or *o* instead to dired the specific subdirectory in a new Dired buffer. After '`find`' has finished (as indicated by a message and the modeline) all Dired commands work as usual.

`find-dired`

Run '`find`' on a directory *dir*, with find arguments *args*, and go into dired-mode on a buffer of the output. The command run (after changing into *dir*) is

```
find . \( args \) -ls
```

`find-name-dired`

Search *dir* recursively for files matching the globbing pattern *pattern*, and run Dired on those files. *pattern* is a shell wildcard (not an Emacs regexp) and need not be quoted. The command run (after changing into *dir*) is

```
    find . -name 'pattern' -ls
```

`find-grep-dired`

Find files in directory *dir* containing a regexp *arg* and start Dired on output. The command run (after changing into *dir*) is

```
find . -exec grep -s arg {} \; -ls
```

Appendix A Dired Internals

This is a short introduction about how Dired's Tree and Mark features work. You are encouraged to read the code (`dired.el`) for more information.

A.1 Tree Dired Internals

In Tree Dired, instead of just one directory, all or part of the directory *tree* starting at the top level directory (the working directory or `default-directory` of the buffer) may be in a Dired buffer. Each file line belongs to exactly one of those subdirectories. After the `ls` program has inserted its output, Dired parses the buffer once to find out where the subdirectory boundaries are and saves them in the variable `dired-subdir-alist`. The beginning of the headerline inserted by `ls` serves as boundary between subdirectories.

Subsequent *`i`* (`dired-maybe-insert-subdir`) commands update this alist and insert the appropriate headerline. Each retrieval of the filename on the current line first extracts the basename (assuming a more or less standard `ls` output format), and then function `dired-current-directory` looks up the current Dired directory in `dired-subdir-alist`. The lookup is keyed on buffer position, as each buffer position is between exactly two subdirectory boundaries. (The end of the buffer serves as an implicit subdirectory boundary.)

`dired-subdir-alist`

Association list of subdirectories and their buffer positions:

`((`*`lastdir`*` . `*`lastmarker`*`) ... (`*`default-directory`*` . `*`firstmarker`*`)).`

The markers point right before the beginning of the line, so that they separate subdirectories adjacent in the buffer. The directories must be in the form returned by `file-name-as-directory`.

`dired-subdir-regexp`

Value: "^. \\([^ \n\r]+\\)\\(:\\)[\n\r]"

Regexp matching a maybe hidden subdirectory line in '`ls -lR`' output. Subexpression 1 is subdirectory proper, no trailing colon. The match starts at the beginning of the line and ends after the end of the line ('`\n`' or '`\r`'). Subexpression 2 must end right before the '`\n`' or `\r`. This is so that Dired can easily check whether a subdirectory is hidden or not: hidden lines end with '`\r`' (*`C-m`*) instead of a newline.

This regexp used to be `"^. \\(/[^\n\r]*\\)\\(:\\)[\n\r]"`, allowing spaces, but disallowing relative filenames (which occur when browsing ls -lR listing in virtual Dired mode, so I changed it).

Note that `"^. \\([^\n\r]+\\)\\(:\\)[\n\r]"` (desirable since it allows both spaces and relative names) will not always work: if you have a file that ends in a colon, its whole line (including permission bits, date etc.) would be mistaken for a subdirectory headerline when parsing '`ls -lR`' output.

`dired-subdir-regexp` is only relevant for parsing '`ls -lR`' output. If Dired inserts subdirectories itself (using `dired-insert-subdir`), they will always be absolute and there is no restriction on the format of filenames, e.g., they can contain spaces.

A.2 Dired Mark Internals

This is a short overview about how marking files and retrieving marked files in Dired works.

`ls` output is indented two spaces two make room for an optional marker character in front of each file line. Marking simply replaces the first space with the marker character, usually `*` or, for deletions, `D`. Indenting just by one would leave the markers adjacent to the permission bits.

`dired-mark-if`

> The macro `dired-mark-if` is used internally to mark files matching certain criteria. It takes two arguments, the *predicate*, a lisp expression evaluating non-nil on file lines to be marked, and *msg*, a message to be displayed while scanning the buffer. *msg* may be nil to suppress the message.

`dired-mark-map`

> To operate on the marked files, all internal Dired functions ultimately call the macro `dired-mark-map`. It takes two arguments, *body* and *arg*, plus an optional argument *show-progress*:
>
> Perform *body* with point somewhere on each marked line (inside a `save-excursion`) and return a list of *body*'s results. If no marked file could be found, execute *body* on the current line.
>
> If *arg* is an integer, use the next *arg* (or previous -*arg*, if *arg*<0) files instead of the marked files. In that case point is dragged along. This is so that commands on the next ARG (instead of the marked) files can be chained easily. Note that for positive ARG point is left on the first file not operated upon, for negative on the last file operated upon
>
> If *arg* is otherwise non-nil, use current file instead.
>
> If optional third argument *show-progress* evaluates to non-nil, we redisplay the Dired buffer after each file is processed. No guarantee is made about the position on the marked line. *body* must ensure this itself if it depends on this. Search starts at the beginning of the buffer, thus the `car` of the list corresponds to the line nearest to the buffer's bottom. This is also true for (positive and negative) integer values of *arg*. The *body* should not be too long as it is expanded four times.

A common case is to retrieve the names of all marked files:

`dired-mark-get-files`

> Return the marked files as list of strings. The list is in the same order as the buffer, that is, the car is the first marked file. Values returned are normally absolute pathnames. Optional argument *localp* equal to `no-dir` means return the filename proper only, with no directory information; any other non-nil value means make them relative to default-directory. Optional second argument *arg* forces use of other files. If *arg* is an integer, use the next *arg* files. If *arg* is otherwise non-nil, use the current file.

Appendix B Known Problems with Dired

There are some problems with Dired that are either not Dired's fault, hard to fix or not worth fixing.

- Renaming directories usually works fine (all affected Dired and file buffers are updated), but moving a directory between different filesystems (those on different hard disks or different partitions) does not work: it creates a plain target file containing the contents of the original directory (inodes and filenames) or fails completely.

  Unfortunately Emacs' builtin function `rename-file` does not give you a clear error message like '`cross-device link attempted`', but rather a spurious `(file-error "Removing old name" "not owner")`, at least in Emacs 18.55.

  On some systems renaming a directory always fails (even within the same filesystem) with the spurious '`not owner`' error.
- If `foo` is a symlink to a non-existing file, `(file-exists-p "foo")` returns nil. Thus, Dired will overwite such (strange) kinds of symlinks without noticing.

  Dired could test both `file-symlink-p` and `file-exists-p`, but this would slow down all file operations to catch a very rare case.
- Copying a directory does not work - it results in a zero-length target file.

  This comes from Emacs' `copy-file` function, not from Dired.

  If you really want to copy a directory (recursively), use '!' and your favorite shell command to do it (e.g. cp -R or cp -r).
- Initial spaces in a filename are not recognized. If I could be sure that all '`ls`' programs insert exactly one space between the time and the filename, I could easily fix this. But '`ls`' programs tend to vary in their amount of white space, and even with one '`ls`' program there is a difference between year and clocktime formats

  ```
  drwxr-xr-x 2 ab027    thp           512 Aug 13 1990  thp/
  drwxr-xr-x 4 ab027    thp           512 Feb  3 21:59 ./
  ```

  If your '`ls`' supports the '`-b`' switch and quotes spaces with that switch, simply add '`b`' to your `dired-listing-switches`. See Section 1.3 [Listing Files in Dired], page 2.

  Spaces anywhere but at the beginning do work.
- In general, only commands that may have targets outside of the current directory tree update other buffers (copy, move and link commands).

  Especially, deletions, (un)compress, chmod/chgrp/chown update only the current buffer.
- Some compress programs make output even if all goes well. Dired takes output as a sign of trouble and assumes that the subprocess failed.

  Redefine function `dired-check-process-checker` suitably to look closer at the generated output. In Emacs 19, the exit status of compress will be checked.
- Aliases like '`rm -i`' for '`rm`' or '`ls -F`' for '`ls`' can cause problems in Dired's (and Emacs') shell command. (Aliases for '`ls`' only matter if you dired wildcards, because only then the shell is used to run '`ls`'.) Csh expands aliases only for interactive shells, which is probably what you want. In Bash, you can achieve this by testing `PS1` in your `~/.bashrc`:

  ```
  # `.bashrc' file
  ```

```
        # this test fails when invoked by rsh
        if [ "${PS1-no}" != "no" ]       # is this an interactive shell?
        then
        . ~/.bash_alias          # if so, source aliases
        fi
```

- Directory names starting with `-` (a minus) may lose when they are to be created or removed. If you care about this, and your rmdir and mkdir understand about `--` meaning end of options, change `emacs-19.el` accordingly.

  In Emacs 19 the `make-directory` and `remove-directory` operations will be builtin, not implemented with '`rmdir`' and '`mkdir`' subprocesses.
- `dired-nstd.el`: This is still buggy. For example, after you've compressed the last file it may not correctly return that file's absolute pathname (`dired-current-directory` erronously returns nil because of markers collapsed during redisplay), ultimately leading to lisp errors.
- The regexp-using `%`-commands get into an endless loop if you specify a regular expression that matches the empty string.
- Function `find-alternate-file` in Emacs 18.57 has a bug that causes *`C-x C-v RET`* (which usually re-visits the current buffer) to fail on Dired buffers. This is fixed in the version in `emacs-19.el`, automatically loaded by Dired.
- It is not possible to resort the Dired buffer without reverting it. That would be hard to implement (and slow to run) given that ls date format would have to be parsed for '`ls -t`' sorting order.

Dired Variable Index

Dired Function Index

Dired Key Index

Dired Concept Index

Table of Contents