Ediff User's Manual

Ediff version 2.70

March 1998

Michael Kifer

1 Introduction

Ediff provides a convenient way for simultaneous browsing through the differences between a pair (or a triple) of files or buffers (which are called '`variants`' for our purposes). The files being compared, file-A, file-B, and file-C (if applicable) are shown in separate windows (side by side, one above the another, or in separate frames), and the differences are highlighted as you step through them. You can also copy difference regions from one buffer to another (and recover old differences if you change your mind).

Another powerful feature is the ability to merge a pair of files into a third buffer. Merging with an ancestor file is also supported. Furthermore, Ediff is equipped with directory-level capabilities that allow the user to conveniently launch browsing or merging sessions on groups of files in two (or three) different directories.

In addition, Ediff can apply a patch to a file and then let you step though both files, the patched and the original one, simultaneously, difference-by-difference. You can even apply a patch right out of a mail buffer, i.e., patches received by mail don't even have to be saved. Since Ediff lets you copy differences between variants, you can, in effect, apply patches selectively (i.e., you can copy a difference region from '`file.orig`' to '`file`', thereby undoing any particular patch that you don't like).

Ediff even understands multi-file patches and can apply them interactively! (Ediff can recognize multi-file patches only if they are in the context format or GNU unified format. All other patches are treated as 1-file patches. Ediff is [hopefully] using the same algorithm as '`patch`' to determine which files need to be patched.)

Ediff is aware of version control, which lets you compare files with their older versions. Ediff also works with remote and compressed files, automatically ftp'ing them over and uncompressing them. See Chapter 6 [Remote and Compressed Files], page 16, for details.

This package builds upon ideas borrowed from Emerge, and several of Ediff's functions are adaptations from Emerge. Although Ediff subsumes and greatly extends Emerge, much of the functionality in Ediff is influenced by Emerge. The architecture and the interface are, of course, drastically different.

2 Major Entry Points

Ediff can be invoked interactively using the following functions, which can be run either from the minibuffer or from the menu bar. In the menu bar, all Ediff's entry points belong to three submenus of the Tools menu: Compare, Merge, and Apply Patch.

`ediff-files`
`ediff`
: Compare two files.

`ediff-buffers`
: Compare two buffers.

`ediff-files3`
`ediff3`
: Compare three files.

`ediff-buffers3`
: Compare three buffers.

`edirs`
`ediff-directories`
: Compare files common to two directories.

`edirs3`
`ediff-directories3`
: Compare files common to three directories.

`edir-revisions`
`ediff-directory-revisions`
: Compare versions of files in a given directory. Ediff selects only the files that are under version control.

`edir-merge-revisions`
`ediff-merge-directory-revisions`
: Merge versions of files in a given directory. Ediff selects only the files that are under version control.

`edir-merge-revisions-with-ancestor`
`ediff-merge-directory-revisions-with-ancestor`
: Merge versions of files in a given directory using other versions as ancestors. Ediff selects only the files that are under version control.

`ediff-windows-wordwise`
: Compare windows word-by-word.

`ediff-windows-linewise`
: Compare windows line-by-line.

`ediff-regions-wordwise`
: Compare regions word-by-word.

`ediff-regions-linewise`
: Compare regions line-by-line.

`ediff-revision`
: Compare versions of the current buffer, if the buffer is visiting a file under version control.

`ediff-patch-file`
`epatch`

Patch a file or multiple files, then compare. If the patch applies to just one file, Ediff will invoke a regular comparison session. If it is a multi-file patch, then a session group interface will be used and the user will be able to patch the files selectively. See Chapter 5 [Session Groups], page 14, for more details.

Note that `ediff-patch-file` will actually use the '`patch`' utility to change the the original files on disk. This is not that dangerous, since you will always have the original contents of the file saved in another file that has the extension '`.orig`'. Furthermore, if the file is under version control, then you can always back out to one of the previous versions (see the section on Version Countrol in Emacs manual).

`ediff-patch-file` is careful about versions control: if the file to be patched is checked in, then Ediff will offer to check it out, because failing to do so may result in the loss of the changes when the file is checked out the next time.

If you don't intend to modify the file via the patch and just want to see what the patch is all about (and decide later), then `ediff-patch-buffer` might be a better choice.

`ediff-patch-buffer`
`epatch-buffer`

Patch a buffer, then compare. The buffer being patched and the file visited by that buffer (if any) is *not* modified. The result of the patch appears in some other buffer that has the name ending with *_patched*.

This function would refuse to apply a multifile patch to a buffer. Use `ediff-patch-file` for that (and when you want the original file to be modified by the '`patch`' utility).

`ediff-merge-files`
`ediff-merge`

Merge two files.

`ediff-merge-files-with-ancestor`
`ediff-merge-with-ancestor`

Like `ediff-merge`, but with a third ancestor file.

`ediff-merge-buffers`

Merge two buffers.

`ediff-merge-buffers-with-ancestor`

Same but with ancestor.

`edirs-merge`
`ediff-merge-directories`

Merge files common to two directories.

`edirs-merge-with-ancestor`
`ediff-merge-directories-with-ancestor`

Same but using files in a third directory as ancestors. If a pair of files doesn't have an ancestor in the ancestor-directory, you will still be able to merge them without the ancestor.

`ediff-merge-revisions`
: Merge two versions of the file visited by the current buffer.

`ediff-merge-revisions-with-ancestor`
: Same but with ancestor.

`ediff-documentation`
: Brings up this manual.

`ediff-show-registry`
`eregistry`
: Brings up Ediff session registry. This feature enables you to quickly find and restart active Ediff sessions.

If you want Ediff to be loaded from the very beginning of your Emacs session, you should put this line in your '~/.emacs' file:

```
(require 'ediff)
```

Otherwise, Ediff will be loaded automatically when you use one of the above functions, either directly or through the menus.

When the above functions are invoked, the user is prompted for all the necessary information—typically the files or buffers to compare, merge, or patch. Ediff tries to be smart about these prompts. For instance, in comparing/merging files, it will offer the visible buffers as defaults. In prompting for files, if the user enters a directory, the previously input file name will be appended to that directory. In addition, if the variable `ediff-use-last-dir` is not `nil`, Ediff will offer previously entered directories as defaults (which will be maintained separately for each type of file, A, B, or C).

All the above functions use the POSIX `diff` or `diff3` programs to find differences between two files. They process the `diff` output and display it in a convenient form. At present, Ediff understands only the plain output from diff. Options such as '`-c`' are not supported, nor is the format produced by incompatible file comparison programs such as the VMS version of `diff`.

The functions `ediff-files`, `ediff-buffers`, `ediff-files3`, `ediff-buffers3` first display the coarse, line-based difference regions, as reported by the '`diff`' program. The total number of difference regions and the current difference number are always displayed in the mode line of the control window.

Since `diff` may report fairly large chunks of text as being different, even though the difference may be localized to just a few words or even to the white space or line breaks, Ediff further *refines* the regions to indicate which exact words differ. If the only difference is in the white space and line breaks, Ediff says so.

On a color display, fine differences are highlighted with color; on a monochrome display, they are underlined. See Section 7.5 [Highlighting Difference Regions], page 22, for information on how to customize this.

The functions `ediff-windows-wordwise`, `ediff-windows-linewise`, `ediff-regions-wordwise` and `ediff-regions-linewise` do comparison on parts of existing Emacs buffers. Since `ediff-windows-wordwise` and `ediff-regions-wordwise` are intended for relatively small segments of buffers, comparison is done on the basis of words rather than lines. No refinement is necessary in this case. These commands are recommended only for relatively small regions (perhaps, up to 100 lines), because these functions have a relatively slow startup.

To compare large regions, use `ediff-regions-linewise`. This command displays differences much like `ediff-files` and `ediff-buffers`.

The functions `ediff-patch-file` and `ediff-patch-buffer` apply a patch to a file or a buffer and then run Ediff on the appropriate files/buffers, displaying the difference regions.

The entry points `ediff-directories`, `ediff-merge-directories`, etc., provide a convenient interface for comparing and merging files in different directories. The user is presented with Dired-like interface from which one can run a group of related Ediff sessions.

For files under version control, `ediff-revision` lets you compare the file visited by the current buffer to one of its checked-in versions. You can also compare two checked-in versions of the visited file. Moreover, the functions `ediff-directory-revisions`, `ediff-merge-directory-revisions`, etc., let you run a group of related Ediff sessions by taking a directory and comparing (or merging) versions of files in that directory.

3 Session Commands

All Ediff commands are displayed in a Quick Help window, unless you type *?* to shrink the window to just one line. You can redisplay the help window by typing *?* again. The Quick Help commands are detailed below.

Many Ediff commands take numeric prefix arguments. For instance, if you type a number, say 3, and then *j* (`ediff-jump-to-difference`), Ediff moves to the third difference region. Typing 3 and then `a` (`ediff-diff-to-diff`) copies the 3d difference region from variant A to variant B. Likewise, 4 followed by *ra* restores the 4th difference region in buffer A (if it was previously written over via the command `a`).

Some commands take negative prefix arguments as well. For instance, typing `-` and then *j* will make the last difference region current. Typing *-2* then *j* makes the penultimate difference region current, etc.

Without the prefix argument, all commands operate on the currently selected difference region. You can make any difference region current using the various commands explained below.

For some commands, the actual value of the prefix argument is immaterial. However, if supplied, the prefix argument may modify the command (see *ga*, *gb*, and *gc*).

3.1 Quick Help Commands

? Toggles the Ediff Quick Help window ON and OFF.

G Prepares a mail buffer for sending a praise or a curse to the Ediff maintainer.

E Brings up the top node of this manual, where you can find further information on the various Ediff functions and advanced issues, such as customization, session groups, etc.

v Scrolls up buffers A and B (and buffer C where appropriate) in a coordinated fashion.

V Scrolls the buffers down.

`<` Scrolls the buffers to the left simultaneously.

`>` Scrolls buffers to the right.

wd Saves the output from the diff utility, for further reference.

With prefix argument, saves the plain output from '`diff`' (see `ediff-diff-program` and `ediff-diff-options`). Without the argument, it saves customized '`diff`' output (see `ediff-custom-diff-program` and `ediff-custom-diff-options`), if it is available.

wa Saves buffer A, if it was modified.

wb Saves buffer B, if it was modified.

wc Saves buffer C, if it was modified (if you are in a session that compares three files simultaneously).

`a`
: *In comparison sessions:* Copies the current difference region (or the region specified as the prefix to this command) from buffer A to buffer B. Ediff saves the old contents of buffer B's region; it can be restored via the command `rb`, which see.

  In merge sessions: Copies the current difference region (or the region specified as the prefix to this command) from buffer A to the merge buffer. The old contents of this region in buffer C can be restored via the command `r`.

`b`
: Works similarly, but copies the current difference region from buffer B to buffer A (in *comparison sessions*) or the merge buffer (in *merge sessions*).

  Ediff saves the old contents of the difference region copied over; it can be reinstated via the command `ra` in comparison sessions and `r` in merge sessions.

`ab`
: Copies the current difference region (or the region specified as the prefix to this command) from buffer A to buffer B. This (and the next five) command is enabled only in sessions that compare three files simultaneously. The old region in buffer B is saved and can be restored via the command `rb`.

`ac`
: Copies the difference region from buffer A to buffer C. The old region in buffer C is saved and can be restored via the command `rc`.

`ba`
: Copies the difference region from buffer B to buffer A. The old region in buffer A is saved and can be restored via the command `ra`.

`bc`
: Copies the difference region from buffer B to buffer C. The command `rc` undoes this.

`ca`
: Copies the difference region from buffer C to buffer A. The command `ra` undoes this.

`cb`
: Copies the difference region from buffer C to buffer B. The command `rb` undoes this.

`p`
`DEL`
: Makes the previous difference region current.

`n`
`SPC`
: Makes the next difference region current.

`j`
`-j`
`Nj`
: Makes the very first difference region current.

  `-j` makes the last region current. Typing a number, N, and then 'j' makes the difference region N current. Typing -N (a negative number) then 'j' makes current the region Last - N.

`ga`
: Makes current the difference region closest to the position of the point in buffer A.

  However, with a prefix argument, Ediff would position all variants around the area indicated by the current point in buffer A: if the point is inside a difference region, then the variants will be positioned at this difference region. If the point is not in any difference region, then it is in an area where all variants agree with each other. In this case, the variants will be positioned so that each would display this area (of agreement).

gb Makes current the difference region closest to the position of the point in buffer B.

With a prefix argument, behaves like `ga`, but with respect to buffer B.

gc *In merge sessions:* makes current the difference region closest to the point in the merge buffer.

In 3-file comparison sessions: makes current the region closest to the point in buffer C.

With a prefix argument, behaves like `ga`, but with respect to buffer C.

! Recomputes the difference regions, bringing them up to date. This is often needed because it is common to do all sorts of editing during Ediff sessions, so after a while, the highlighted difference regions may no longer reflect the actual differences among the buffers.

*** Forces refinement of the current difference region, which highlights the exact words of disagreement among the buffers. With a negative prefix argument, unhighlights the current region.

Forceful refinement may be needed if Ediff encounters a difference region that is larger than `ediff-auto-refine-limit`. In this situation, Ediff doesn't do automatic refinement in order to improve response time. (Ediff doesn't auto-refine on dumb terminals as well, but `*` still works there. However, the only useful piece of information it can tell you is whether or not the difference regions disagree only in the amount of white space.)

This command is also useful when the highlighted fine differences are no longer current, due to user editing.

m Displays the current Ediff session in a frame as wide as the physical display. This is useful when comparing files side-by-side. Typing 'm' again restores the original size of the frame.

| Toggles the horizontal/vertical split of the Ediff display. Horizontal split is convenient when it is possible to compare files side-by-side. If the frame in which files are displayed is too narrow and lines are cut off, typing *m* may help some.

@ Toggles auto-refinement of difference regions (i.e., automatic highlighting of the exact words that differ among the variants). Auto-refinement is turned off on devices where Emacs doesn't support highlighting.

On slow machines, it may be advantageous to turn auto-refinement off. The user can always forcefully refine specific difference regions by typing `*`.

h Cycles between full highlighting, the mode where fine differences are not highlighted (but computed), and the mode where highlighting is done with ASCII strings. The latter is not really recommended, unless on a dumb TTY.

r Restores the old contents of the region in the merge buffer. (If you copied a difference region from buffer A or B into the merge buffer using the commands `a` or *b*, Ediff saves the old contents of the region in case you change your mind.)

This command is enabled in merge sessions only.

ra Restores the old contents of the current difference region in buffer A, which was previously saved when the user invoked one of these commands: *b*, *ba*, *ca*, which see. This command is enabled in comparison sessions only.

rb Restores the old contents of the current difference region in buffer B, which was previously saved when the user invoked one of these commands: *a*, *ab*, *cb*, which see. This command is enabled in comparison sessions only.

rc Restores the old contents of the current difference region in buffer C, which was previously saved when the user invoked one of these commands: *ac*, *bc*, which see. This command is enabled in 3-file comparison sessions only.

Tell Ediff to skip over regions that disagree among themselves only in the amount of white space and line breaks.

Even though such regions will be skipped over, you can still jump to any one of them by typing the region number and then 'j'. Typing *##* again puts Ediff back in the original state.

#h
#f Ediff works hard to ameliorate the effects of boredom in the workplace...

Quite often differences are due to identical replacements (e.g., the word 'foo' is replaced with the word 'bar' everywhere). If the number of regions with such boring differences exceeds your tolerance threshold, you may be tempted to tell Ediff to skip these regions altogether (you will still be able to jump to them via the command *j*). The above commands, *#h* and *#f*, may well save your day!

#h prompts you to specify regular expressions for each variant. Difference regions where each variant's region matches the corresponding regular expression will be skipped from then on. (You can also tell Ediff to skip regions where at least one variant matches its regular expression.)

#f does dual job: it focuses on regions that match the corresponding regular expressions. All other regions will be skipped over. See Section 7.4 [Selective Browsing], page 21, for more.

A Toggles the read-only property in buffer A. If file A is under version control and is checked in, it is checked out (with your permission).

B Toggles the read-only property in buffer B. If file B is under version control and is checked in, it is checked out.

C Toggles the read-only property in buffer C (in 3-file comparison sessions). If file C is under version control and is checked in, it is checked out.

~ Swaps the windows where buffers A and B are displayed. If you are comparing three buffers at once, then this command would rotate the windows among buffers A, B, and C.

i Displays all kinds of useful data about the current Ediff session.

D Runs `ediff-custom-diff-program` on the variants and displays the buffer containing the output. This is useful when you must send the output to your Mom.

With a prefix argument, displays the plain '`diff`' output. See Section 7.8 [Patch and Diff Programs], page 25, for details.

R Displays a list of currently active Ediff sessions—the Ediff Registry. You can then restart any of these sessions by either clicking on a session record or by putting the cursor over it and then typing the return key.

(Some poor souls leave so many active Ediff sessions around that they loose track of them completely... The 'R' command is designed to save these people from the recently discovered Ediff Proficiency Syndrome.)

Typing *R* brings up Ediff Registry only if it is typed into an Ediff Control Panel. If you don't have a control panel handy, type this in the minibuffer: *M-x eregistry*. See Chapter 4 [Registry of Ediff Sessions], page 13.

M Shows the session group buffer that invoked the current Ediff session. See Chapter 5 [Session Groups], page 14, for more information on session groups.

z Suspends the current Ediff session. (If you develop a condition known as Repetitive Ediff Injury—a serious but curable illness—you must change your current activity. This command tries hard to hide all Ediff-related buffers.)

The easiest way to resume a suspended Ediff session is through the registry of active sessions. See Chapter 4 [Registry of Ediff Sessions], page 13, for details.

q Terminates this Ediff session. With a prefix argument (e.g.,*1q*), asks if you also want to delete the buffers of the variants. Modified files and the results of merges are never deleted.

% Toggles narrowing in Ediff buffers. Ediff buffers may be narrowed if you are comparing only parts of these buffers via the commands `ediff-windows-*` and `ediff-regions-*`, which see.

C-l Restores the usual Ediff window setup. This is the quickest way to resume an Ediff session, but it works only if the control panel of that session is visible.

$ While merging with an ancestor file, Ediff is determined to reduce user's wear and tear by saving him and her much of unproductive, repetitive typing. If it notices that, say, file A's difference region is identical to the same difference region in the ancestor file, then the merge buffer will automatically get the difference region taken from buffer B. The rationale is that this difference region in buffer A is as old as that in the ancestor buffer, so the contents of that region in buffer B represents real change.

You may want to ignore such 'obvious' merges and concentrate on difference regions where both files 'clash' with the ancestor, since this means that two different people have been changing this region independently and they had different ideas on how to do this.

The above command does this for you by skipping the regions where only one of the variants clashes with the ancestor but the other variant agrees with it. Typing *$* again undoes this setting.

/ Displays the ancestor file during merges.

`&` In some situations, such as when one of the files agrees with the ancestor file on a difference region and the other doesn't, Ediff knows what to do: it copies the current difference region from the second buffer into the merge buffer.

In other cases, the right course of action is not that clearcut, and Ediff would use a default action. The above command changes the default action. The default action can be '`default-A`' (choose the region from buffer A), '`default-B`' (choose the region from buffer B), or '`combined`' (combine the regions from the two buffers). See Section 7.9 [Merging and diff3], page 27, for further details.

The command `&` also affects the regions in the merge buffers that have '`default-A`', '`default-B`', or '`combined`' status, provided they weren't changed with respect to the original. For instance, if such a region has the status '`default-A`' then changing the default action to '`default-B`' will also replace this merge-buffer's region with the corresponding region from buffer B.

`s` Causes the merge window shrink to its minimum size, thereby exposing as much of the variant buffers as possible. Typing 's' again restores the original size of that window.

With a positive prefix argument, this command enlarges the merge window. E.g., `4s` increases the size of the window by about 4 lines, if possible. With a negative numeric argument, the size of the merge window shrinks by that many lines, if possible. Thus, `-s` shrinks the window by about 1 line and `-3s` by about 3 lines.

This command is intended only for temporary viewing; therefore, Ediff restores window C to its original size whenever it makes any other change in the window configuration. However, redisplaying (`C-l`) or jumping to another difference does not affect window C's size.

The split between the merge window and the variant windows is controlled by the variable `ediff-merge-window-share`, which see.

`+` Combines the difference regions from buffers A and B and copies the result into the merge buffer. See Section 7.9 [Merging and diff3], page 27 and the variables `ediff-combine-diffs` and `ediff-combination-pattern`.

`=` You may run into situations when a large chunk of text in one file has been edited and then moved to a different place in another file. In such a case, these two chunks of text are unlikely to belong to the same difference region, so the refinement feature of Ediff will not be able to tell you what exactly differs inside these chunks. Since eyeballing large pieces of text is contrary to human nature, Ediff has a special command to help reduce the risk of developing a cataract.

The above command compares regions within Ediff buffers. This creates a child Ediff session for comparing current Emacs regions in buffers A, B, or C as follows:

If you are comparing 2 files or buffers: Ediff would compare current Emacs regions in buffers A and B.

If you are comparing 3 files or buffers simultaneously: Ediff would compare the current Emacs regions in the buffers of your choice (you will be asked which two of the three buffers to use).

If you are merging files or buffers (with or without ancestor): Ediff would take the current region in the merge buffer and compare it to the current region in the buffer of your choice (A or B).

Highlighting set by the parent Ediff session is removed, to avoid interference with highlighting of the child session. When done with the child session, type `C-l` in the parent's control panel to restore the original highlighting.

If you temporarily switch to the parent session, parent highlighting will be restored. If you then come back to the child session, you may want to remove parent highlighting, so it won't interfere. Typing `h` may help here.

3.2 Other Session Commands

The following commands can be invoked from within any Ediff session, although some of them are not bound to a key.

`eregistry`
`ediff-show-registry`
: This command brings up the registry of active Ediff sessions. Ediff registry is a device that can be used to resume any active Ediff session (which may have been postponed because the user switched to some other activity). This command is also useful for switching between multiple active Ediff sessions that are run at the same time. The function `eregistry` is an alias for `ediff-show-registry`. See Chapter 4 [Registry of Ediff Sessions], page 13, for more information on this registry.

`ediff-toggle-multiframe`
: Changes the display from the multi-frame mode (where the quick help window is in a separate frame) to the single-frame mode (where all Ediff buffers share the same frame), and vice versa. See `ediff-window-setup-function` for details on how to make either of these modes the default one.

  This function can also be invoked from the Menubar. However, in some cases, the change will take place only after you execute one of the Ediff commands, such as going to the next difference or redisplaying.

`ediff-revert-buffers-then-recompute-diffs`
: This command reverts the buffers you are comparing and recomputes their differences. It is useful when, after making changes, you decided to make a fresh start, or if at some point you changed the files being compared but want to discard any changes to comparison buffers that were done since then.

  This command normally asks for confirmation before reverting files. With a prefix argument, it reverts files without asking.

`ediff-profile`
: Ediff has an admittedly primitive (but useful) facility for profiling Ediff's commands. It is meant for Ediff maintenance—specifically, for making it run faster. The function `ediff-profile` toggles profiling of ediff commands.

4 Registry of Ediff Sessions

Ediff maintains a registry of all its invocations that are still *active*. This feature is very convenient for switching among active Ediff sessions or for quickly restarting a suspended Ediff session.

The focal point of this activity is a buffer called **Ediff Registry**. You can display this buffer by typing `R` in any Ediff Control Buffer or Session Group Buffer (see Chapter 5 [Session Groups], page 14), or by typing `M-x eregistry` into the Minibuffer. The latter would be the fastest way to bring up the registry buffer if no control or group buffer is displayed in any of the visible Emacs windows. If you are in a habit of running multiple long Ediff sessions and often need to suspend, resume, or switch between them, it may be a good idea to have the registry buffer permanently displayed in a separate, dedicated window.

The registry buffer has several convenient key bindings. For instance, clicking mouse button 2 or typing `RET` or `v` over any session record resumes that session. Session records in the registry buffer provide a fairly complete description of each session, so it is usually easy to identify the right session to resume.

Other useful commands are bound to `SPC` (next registry record) and `DEL` (previous registry record). There are other commands as well, but you don't need to memorize them, since they are listed at the top of the registry buffer.

5 Session Groups

Several major entries of Ediff perform comparison and merging on directories. On entering `ediff-directories`, `ediff-directories3`, `ediff-merge-directories`, `ediff-merge-directories-with-ancestor`, `ediff-directory-revisions`, `ediff-merge-directory-revisions`, or `ediff-merge-directory-revisions-with-ancestor`, the user is presented with a Dired-like buffer that lists files common to the directories involved along with their sizes. (The list of common files can be further filtered through a regular expression, which the user is prompted for.) We call this buffer *Session Group Panel* because all Ediff sessions associated with the listed files will have this buffer as a common focal point.

Clicking button 2 or typing *RET* or *v* over a record describing files invokes Ediff in the appropriate mode on these files. You can come back to the session group buffer associated with a particular invocation of Ediff by typing *M* in Ediff control buffer of that invocation.

Many commands are available in the session group buffer; some are applicable only to certain types of work. The relevant commands are always listed at the top of each session group buffer, so there is no need to memorize them.

In directory comparison or merging, a session group panel displays only the files common to all directories involved. The differences are kept in a separate buffer and are conveniently displayed by typing *D* to the corresponding session group panel. Thus, as an added benefit, Ediff can be used to compare the contents of up to three directories.

Session records in session group panels are also marked with `+`, for active sessions, and with `-`, for finished sessions.

Sometimes, it is convenient to exclude certain sessions from a group. Usually this happens when the user doesn't intend to run Ediff of certain files in the group, and the corresponding session records just add clutter to the session group buffer. To help alleviate this problem, the user can type *h* to mark a session as a candidate for exclusion and *x* to actually hide the marked sessions. There actions are reversible: with a prefix argument, *h* unmarks the session under the cursor, and *x* brings the hidden sessions into the view (*x* doesn't unmark them, though, so the user has to explicitly unmark the sessions of interest).

Group sessions also understand the command *m*, which marks sessions for future operations (other than hiding) on a group of sessions. At present, the only such group-level operation is the creation of a multi-file patch.

For group sessions created to merge files, Ediff can store all merges automatically in a directory. The user is asked to specify such directory if the value of `ediff-autostore-merges` is non-nil. If the value is `nil`, nothing is done to the merge buffers—it will be the user's responsibility to save them. If the value is `t`, the user will be asked where to save the merge buffers in all merge jobs, even those that do not originate from a session group. It the value is neither `nil` nor `t`, the merge buffer is saved *only* if this merge session was invoked from a session group. This behavior is implemented in the function `ediff-maybe-save-and-delete-merge`, which is a hook in `ediff-quit-merge-hook`. The user can supply a different hook, if necessary.

The variable `ediff-autostore-merges` is buffer-local, so it can be set in a per-buffer manner. Therefore, use `setq-default` to globally change this variable.

A multi-file patch is a concatenated output of several runs of the Unix '`diff`' command (some versions of '`diff`' let you create a multi-file patch in just one run). Ediff facilitates creation of multi-file patches as follows. If you are in a session group buffer created in response to `ediff-directories` or `ediff-directory-revisions`, you can mark (by typing `m`) the desired Ediff sessions and then type `P` to create a multi-file patch of those marked sessions. Ediff will then display a buffer containing the patch. The patch is generated by invoking '`diff`' on all marked individual sessions (represented by files) and session groups (represented by directories). Ediff will also recursively descend into any *unmarked* session group and will search for marked sessions there. In this way, you can create multi-file patches that span file subtrees that grow out of any given directory.

In an `ediff-directories` session, it is enough to just mark the requisite sessions. In `ediff-directory-revisions` revisions, the marked sessions must also be active, or else Ediff will refuse to produce a multi-file patch. This is because, in the latter-style sessions, there are many ways to create diff output, and it is easier to handle by running Ediff on the inactive sessions.

Last, but not least, by typing `=`, you can quickly find out which sessions have identical files, so you won't have to run Ediff on those sessions. This, however, works only on local, uncompressed files. For compressed or remote files, this command won't report anything.

6 Remote and Compressed Files

Ediff works with remote, compressed, and encrypted files. Ediff supports 'ange-ftp.el', 'jka-compr.el', 'uncompress.el' and 'crypt++.el', but it may work with other similar packages as well. This means that you can compare files residing on another machine, or you can apply a patch to a file on another machine. Even the patch itself can be a remote file!

When patching compressed or remote files, Ediff does not rename the source file (unlike what the `patch` utility would usually do). Instead, the source file retains its name and the result of applying the patch is placed in a temporary file that has the suffix '_patched' attached. Generally, this applies to files that are handled using black magic, such as special file handlers (ange-ftp and some compression and encryption packages also use this method).

Regular files are treated by the `patch` utility in the usual manner, i.e., the original is renamed into 'source-name.orig' and the result of the patch is placed into the file source-name ('_orig' is used on systems like VMS, DOS, etc.)

7 Customization

Ediff has a rather self-explanatory interface, and in most cases you won't need to change anything. However, should the need arise, there are extensive facilities for changing the default behavior.

Most of the customization can be done by setting various variables in the '`.emacs`' file. Some customization (mostly window-related customization and faces) can be done by putting appropriate lines in '`.Xdefaults`', '`.xrdb`', or whatever X resource file is in use.

With respect to the latter, please note that the X resource for Ediff customization is 'Ediff', *not* 'emacs'. See Section 7.3 [Window and Frame Configuration], page 19, See Section 7.5 [Highlighting Difference Regions], page 22, for further details. Please also refer to Emacs manual for the information on how to set Emacs X resources.

7.1 Hooks

The bulk of customization can be done via the following hooks:

`ediff-load-hook`
: This hook can be used to change defaults after Ediff is loaded.

`ediff-keymap-setup-hook`
: This hook can be used to alter bindings in Ediff's keymap, `ediff-mode-map`. These hooks are run right after the default bindings are set but before `ediff-load-hook`. The regular user needs not be concerned with this hook—it is provided for implementors of other Emacs packages built on top of Ediff.

`ediff-before-setup-windows-hook`
`ediff-after-setup-windows-hook`
: These two hooks are called before and after Ediff sets up its window configuration. Can be used to save the configuration that existed before Ediff starts or for whatever other purposes.

`ediff-suspend-hook`
`ediff-quit-hook`
: These two hooks are run when you suspend or quit Ediff. They can be used to set desired window configurations, delete files Ediff didn't want to clean up after exiting, etc.

  By default, `ediff-quit-hook` holds one hook function, `ediff-cleanup-mess`, which cleans after Ediff, as appropriate in most cases. You probably won't want to change it, but you might want to add other hook functions.

  Keep in mind that hooks executing before `ediff-cleanup-mess` start in `ediff-control-buffer;` they should also leave `ediff-control-buffer` as the current buffer when they finish. Hooks that are executed after `ediff-cleanup-mess` should expect the current buffer be either buffer A or buffer B. `ediff-cleanup-mess` doesn't kill the buffers being compared or merged (see `ediff-cleanup-hook`, below).

`ediff-cleanup-hook`
: This hook is run just before `ediff-quit-hook`. This is a good place to do various cleanups, such as deleting the variant buffers. Ediff provides a function, `ediff-janitor`, as one such possible hook, which you can add to `ediff-cleanup-hook` with `add-hooks`.

  This function kills buffers A, B, and, possibly, C, if these buffers aren't modified. In merge jobs, buffer C is never deleted. However, the side effect of using this function is that you may not be able to compare the same buffer in two separate Ediff sessions: quitting one of them will delete this buffer in another session as well.

`ediff-quit-merge-hook`
: This hook is called when Ediff quits a merge job. By default, the value is `ediff-maybe-save-and-delete-merge`, which is a function that attempts to save the merge buffer according to the value of `ediff-autostore-merges`, as described later.

`ediff-before-setup-control-frame-hook`
`ediff-after-setup-control-frame-hook`
: These two hooks run before and after Ediff sets up the control frame. They can be used to relocate Ediff control frame when Ediff runs in a multiframe mode (i.e., when the control buffer is in its own dedicated frame). Be aware that many variables that drive Ediff are local to Ediff Control Panel (`ediff-control-buffer`), which requires special care in writing these hooks. Take a look at `ediff-default-suspend-hook` and `ediff-default-quit-hook` to see what's involved.

`ediff-startup-hook`
: This hook is run at the end of Ediff startup.

`ediff-select-hook`
: This hook is run after Ediff selects the next difference region.

`ediff-unselect-hook`
: This hook is run after Ediff unselects the current difference region.

`ediff-prepare-buffer-hook`
: This hook is run for each Ediff buffer (A, B, C) right after the buffer is arranged.

`ediff-display-help-hook`
: Ediff runs this hook each time after setting up the help message. It can be used to alter the help message for custom packages that run on top of Ediff.

`ediff-mode-hook`
: This hook is run just after Ediff mode is set up in the control buffer. This is done before any Ediff window is created. You can use it to set local variables that alter the look of the display.

`ediff-registry-setup-hook`
: Hooks run after setting up the registry for all active Ediff session. See Chapter 5 [Session Groups], page 14, for details.

`ediff-session-group-setup-hook`
: Hooks run after setting up a control panel for a group of related Ediff sessions. See Chapter 5 [Session Groups], page 14, for details.

`ediff-quit-session-group-hook`
: Hooks run just before exiting a session group.

`ediff-meta-buffer-keymap-setup-hook`
: Hooks run just after setting up the `ediff-meta-buffer-map` — the map that controls key bindings in the meta buffer. Since `ediff-meta-buffer-map` is a local variable, you can set different bindings for different kinds of meta buffers.

7.2 Quick Help Customization

Ediff provides quick help using its control panel window. Since this window takes a fair share of the screen real estate, you can toggle it off by typing *?*. The control window will then shrink to just one line and a mode line, displaying a short help message.

The variable `ediff-use-long-help-message` tells Ediff whether you use the short message or the long one. By default, it is set to `nil`, meaning that the short message is used. Set this to `t`, if you want Ediff to use the long message by default. This property can always be changed interactively, by typing *?* into Ediff Control Buffer.

If you want to change the appearance of the help message on a per-buffer basis, you must use `ediff-startup-hook` to change the value of the variable `ediff-help-message`, which is local to `ediff-control-buffer`.

7.3 Window and Frame Configuration

On a non-windowing display, Ediff sets things up in one frame, splitting it between a small control window and the windows for buffers A, B, and C. The split between these windows can be horizontal or vertical, which can be changed interactively by typing `|` while the cursor is in the control window.

On a window display, Ediff sets up a dedicated frame for Ediff Control Panel and then it chooses windows as follows: If one of the buffers is invisible, it is displayed in the currently selected frame. If a buffer is visible, it is displayed in the frame where it is visible. If, according to the above criteria, the two buffers fall into the same frame, then so be it—the frame will be shared by the two. The same algorithm works when you type *C-l* (`ediff-recenter`), *p* (`ediff-previous-difference`), *n* (`ediff-next-difference`), etc.

The above behavior also depends on whether the current frame is splittable, dedicated, etc. Unfortunately, the margin of this book is too narrow to present the details of this remarkable algorithm.

The upshot of all this is that you can compare buffers in one frame or in different frames. The former is done by default, while the latter can be achieved by arranging buffers A, B (and C, if applicable) to be seen in different frames. Ediff respects these arrangements, automatically adapting itself to the multi-frame mode.

Ediff uses the following variables to set up its control panel (a.k.a. control buffer, a.k.a. quick help window):

`ediff-control-frame-parameters`
: You can change or augment this variable including the font, color, etc. The X resource name of Ediff Control Panel frames is '`Ediff`'. Under X-windows, you can use this name to set up preferences in your '`~/.Xdefaults`', '`~/.xrdb`', or whatever X resource file is in use. Usually this is preferable to changing `ediff-control-frame-parameters` directly. For instance, you can specify in '`~/.Xdefaults`' the color of the control frame using the resource '`Ediff*background`'.

  In general, any X resource pertaining the control frame can be reached via the prefix `Ediff*`.

`ediff-control-frame-position-function`
: The preferred way of specifying the position of the control frame is by setting the variable `ediff-control-frame-position-function` to an appropriate function. The default value of this variable is `ediff-make-frame-position`. This function places the control frame in the vicinity of the North-East corner of the frame displaying buffer A.

The following variables can be used to adjust the location produced by `ediff-make-frame-position` and for related customization.

`ediff-narrow-control-frame-leftward-shift`
: Specifies the number of characters for shifting the control frame from the rightmost edge of frame A when the control frame is displayed as a small window.

`ediff-wide-control-frame-rightward-shift`
: Specifies the rightward shift of the control frame from the left edge of frame A when the control frame shows the full menu of options.

`ediff-control-frame-upward-shift`
: Specifies the number of pixels for the upward shift of the control frame.

`ediff-prefer-iconified-control-frame`
: If this variable is `t`, the control frame becomes iconified automatically when you toggle the quick help message off. This saves valuable real estate on the screen. Toggling help back will deiconify the control frame.

  To start Ediff with an iconified Control Panel, you should set this variable to `t` and `ediff-prefer-long-help-message` to `nil` (see Section 7.2 [Quick Help Customization], page 19). This behavior is useful only if the window manager is TWM or a derivative.

To make more creative changes in the way Ediff sets up windows, you can rewrite the function `ediff-setup-windows`. However, we believe that detaching Ediff Control Panel from the rest and making it into a separate frame offers an important opportunity by allowing you to iconify that frame. The icon will usually accept all of the Ediff commands, but will free up valuable real estate on your screen (this may depend on your window manager, though).

The following variable controls how windows are set up:

`ediff-window-setup-function`

The multiframe setup is done by the `ediff-setup-windows-multiframe` function, which is the default on windowing displays. The plain setup, one where all windows are always in one frame, is done by `ediff-setup-windows-plain`, which is the default on a non-windowing display (or in an xterm window). In fact, under Emacs, you can switch freely between these two setups by executing the command `ediff-toggle-multiframe` using the Minibuffer of the Menubar.

If you don't like any of these setups, write your own function. See the documentation for `ediff-window-setup-function` for the basic guidelines. However, writing window setups is not easy, so you should first take a close look at `ediff-setup-windows-plain` and `ediff-setup-windows-multiframe`.

You can run multiple Ediff sessions at once, by invoking Ediff several times without exiting previous Ediff sessions. Different sessions may even operate on the same pair of files.

Each session has its own Ediff Control Panel and all the regarding a particular session is local to the associated control panel buffer. You can switch between sessions by suspending one session and then switching to another control panel. (Different control panel buffers are distinguished by a numerical suffix, e.g., '`Ediff Control Panel<3>`'.)

7.4 Selective Browsing

Sometimes it is convenient to be able to step through only some difference regions, those that match certain regular expressions, and to ignore all others. On other occasions, you may want to ignore difference regions that match some regular expressions, and to look only at the rest.

The commands *#f* and *#h* let you do precisely this.

Typing *#f* lets you specify regular expressions that match difference regions you want to focus on. We shall call these regular expressions *regexp-A*, *regexp-B* and *regexp-C*. Ediff will then start stepping through only those difference regions where the region in buffer A matches *regexp-A* and/or the region in buffer B matches *regexp-B*, etc. Whether 'and' or 'or' will be used depends on how you respond to a question.

When scanning difference regions for the aforesaid regular expressions, Ediff narrows the buffers to those regions. This means that you can use the expressions *\`* and *\'* to tie search to the beginning or end of the difference regions.

On the other hand, typing *#h* lets you specify (hide) uninteresting regions. That is, if a difference region in buffer A matches *regexp-A*, the corresponding region in buffer B matches *regexp-B* and (if applicable) buffer C's region matches *regexp-C*, then the region will be ignored by the commands *n*/⟨SPC⟩ (`ediff-next-difference`) and *p*/⟨DEL⟩ (`ediff-previous-difference`) commands.

Typing *#f* and *#h* toggles selective browsing on and off.

Note that selective browsing affects only `ediff-next-difference` and `ediff-previous-difference`, i.e., the commands *n*/⟨SPC⟩ and *p*/⟨DEL⟩. *#f* and *#h* do not change the position of the point in the buffers. And you can still jump directly (using *j*) to any numbered difference.

Users can supply their own functions to specify how Ediff should do selective browsing. To change the default Ediff function, add a function to `ediff-load-hook` which will do the following assignments:

```
(setq ediff-hide-regexp-matches-function 'your-hide-function)
(setq ediff-focus-on-regexp-matches-function 'your-focus-function)
```

Useful hint: To specify a regexp that matches everything, don't simply type ⟨RET⟩ in response to a prompt. Typing ⟨RET⟩ tells Ediff to accept the default value, which may not be what you want. Instead, you should enter something like ⟨^⟩ or ⟨$⟩. These match every line.

You can use the status command, *i*, to find out whether selective browsing is currently in effect.

The regular expressions you specified are kept in the local variables `ediff-regexp-focus-A`, `ediff-regexp-focus-B`, `ediff-regexp-focus-C`, `ediff-regexp-hide-A`, `ediff-regexp-hide-B`, `ediff-regexp-hide-C`. Their default value is the empty string (i.e., nothing is hidden or focused on). To change the default, set these variables in '`.emacs`' using `setq-default`.

In addition to the ability to ignore regions that match regular expressions, Ediff can be ordered to start skipping over certain "uninteresting" difference regions. This is controlled by the following variable:

`ediff-ignore-similar-regions`
: If `t`, causes Ediff to skip over "uninteresting" difference regions, which are the regions where the variants differ only in the amount of the white space and newlines. This feature can be toggled on/off interactively, via the command `##`.

Note: In order for this feature to work, auto-refining of difference regions must be on, since otherwise Ediff won't know if there are fine differences between regions. On devices where Emacs can display faces, auto-refining is a default, but it is not turned on by default on text-only terminals. In that case, you must explicitly turn auto-refining on (such as, by typing `@`).

Reassurance: If many such uninteresting regions appear in a row, Ediff may take a long time to skip over them because it has to compute fine differences of all intermediate regions. This delay does not indicate any problem.

7.5 Highlighting Difference Regions

The following variables control the way Ediff highlights difference regions:

`ediff-before-flag-bol`
`ediff-after-flag-eol`
`ediff-before-flag-mol`
`ediff-after-flag-mol`
: These variables hold strings that Ediff uses to mark the beginning and the end of the differences found in files A, B, and C on devices where Emacs cannot display faces. Ediff uses different flags to highlight regions that begin/end at the beginning/end of a line or in a middle of a line.

`ediff-current-diff-face-A`
`ediff-current-diff-face-B`
`ediff-current-diff-face-C`
: Ediff uses these faces to highlight current differences on devices where Emacs can display faces. These and subsequently described faces can be set either in '`.emacs`' or in '`.Xdefaults`'. The X resource for Ediff is '`Ediff`', *not* '`emacs`'. Please refer to Emacs manual for the information on how to set X resources.

`ediff-fine-diff-face-A`
`ediff-fine-diff-face-B`
`ediff-fine-diff-face-C`
: Ediff uses these faces to show the fine differences between the current differences regions in buffers A, B, and C, respectively.

`ediff-even-diff-face-A`
`ediff-even-diff-face-B`
`ediff-even-diff-face-C`
`ediff-odd-diff-face-A`
`ediff-odd-diff-face-B`
`ediff-odd-diff-face-C`
: Non-current difference regions are displayed using these alternating faces. The odd and the even faces are actually identical on monochrome displays, because without colors options are limited. So, Ediff uses italics to highlight non-current differences.

`ediff-force-faces`
: Ediff generally can detect when Emacs is running on a device where it can use highlighting with faces. However, if it fails to determine that faces can be used, the user can set this variable to `t` to make sure that Ediff uses faces to highlight differences.

`ediff-highlight-all-diffs`
: Indicates whether—on a windowind display—Ediff should highlight differences using inserted strings (as on text-only terminals) or using colors and highlighting. Normally, Ediff highlights all differences, but the selected difference is highlighted more visibly. One can cycle through various modes of highlighting by typing *h*. By default, Ediff starts in the mode where all difference regions are highlighted. If you prefer to start in the mode where unselected differences are not highlighted, you should set `ediff-highlight-all-diffs` to `nil`. Type *h* to restore highlighting for all differences.

  Ediff lets you switch between the two modes of highlighting. That is, you can switch interactively from highlighting using faces to highlighting using string flags, and back. Of course, switching has effect only under a windowing system. On a text-only terminal or in an xterm window, the only available option is highlighting with strings.

If you want to change the default settings for `ediff-force-faces` and `ediff-highlight-all-diffs`, you must do it **before** Ediff is loaded.

You can also change the defaults for the faces used to highlight the difference regions. There are two ways to do this. The simplest and the preferred way is to use the customization widget accessible from the menubar. Ediff's customization group is located under "Tools", which in turn is under "Programming". The faces that are used to highlight difference regions are located in the "Highlighting" subgroup of the Ediff customization group.

The second, much more arcane, method to change default faces is to include some Lisp code in '`~/.emacs`'. For instance,

```
(setq ediff-current-diff-face-A
      (copy-face 'bold-italic 'ediff-current-diff-face-A))
```

would use the pre-defined fase `bold-italic` to highlight the current difference region in buffer A (this face is not a good choice, by the way).

If you are unhappy with just *some* of the aspects of the default faces, you can modify them when Ediff is being loaded using `ediff-load-hook`. For instance:

```
(add-hook 'ediff-load-hook
          (function (lambda ()
                      (set-face-foreground
                        ediff-current-diff-face-B "blue")
                      (set-face-background
                        ediff-current-diff-face-B "red")
                      (make-face-italic
                        ediff-current-diff-face-B))))
```

Note: it is not recommended to use `internal-get-face` when defining Ediff's faces, since this may cause problems when there are several frames with different font sizes. Instead, use `copy-face` or `set/make-face-...` as shown above.

7.6 Narrowing

If buffers being compared are narrowed at the time of invocation of Ediff, `ediff-buffers` will preserve the narrowing range. However, if `ediff-files` is invoked on the files visited by these buffers, that would widen the buffers, since this command is defined to compare the entire files.

Calling `ediff-regions-linewise` or `ediff-windows-linewise`, or the corresponding '`-wordwise`' commands, narrows the variants to the particular regions being compared. The original accessible ranges are restored when you quit Ediff. During the command, you can toggle this narrowing on and off with the `%` command.

These two variables control this narrowing behavior:

`ediff-start-narrowed`
: If `t`, Ediff narrows the display to the appropriate range when it is invoked with an '`ediff-regions...`' or '`ediff-windows...`' command. If `nil`, these commands do not automatically narrow, but you can still toggle narrowing on and off by typing `%`.

`ediff-quit-widened`
: Controls whether on quitting Ediff should restore the accessible range that existed before the current invocation.

7.7 Refinement of Difference Regions

Ediff has variables to control the way fine differences are highlighted. This feature gives you control over the process of refinement. Note that refinement ignores spaces, tabs, and newlines.

`ediff-auto-refine`
: This variable controls whether fine differences within regions are highlighted automatically ("auto-refining"). The default is yes ('`on`').

  On a slow machine, automatic refinement may be painful. In that case, you can turn auto-refining on or off interactively by typing `@`. You can also turn off display of refining that has already been done.

  When auto-refining is off, fine differences are shown only for regions for which these differences have been computed and saved before. If auto-refining and display of refining are both turned off, fine differences are not shown at all.

  Typing `*` computes and displays fine differences for the current difference region, regardless of whether auto-refining is turned on.

`ediff-auto-refine-limit`
: If auto-refining is on, this variable limits the size of the regions to be auto-refined. This guards against the possible slowdown that may be caused by extraordinary large difference regions.

  You can always refine the current region by typing `*`.

`ediff-forward-word-function`
: This variable controls how fine differences are computed. The value must be a Lisp function that determines how the current difference region should be split into words.

  Fine differences are computed by first splitting the current difference region into words and then passing the result to `ediff-diff-program`. For the default forward word function (which is `ediff-forward-word`), a word is a string consisting of letters, '`-`', or '`_`'; a string of punctuation symbols; a string of digits, or a string consisting of symbols that are neither space, nor a letter.

  This default behavior is controlled by four variables: `ediff-word-1`, ..., `ediff-word-4`. See the on-line documentation for these variables and for the function `ediff-forward-word` for an explanation of how to modify these variables.

Sometimes, when a region has too many differences between the variants, highlighting of fine differences is inconvenient, especially on color displays. If that is the case, type `*` with a negative prefix argument. This unhighlights fine differences for the current region.

To unhighlight fine differences in all difference regions, use the command `@`. Repeated typing of this key cycles through three different states: auto-refining, no-auto-refining, and no-highlighting of fine differences.

7.8 Patch and Diff Programs

This section describes variables that specify the programs to be used for applying patches and for computing the main difference regions (not the fine difference regions):

`ediff-diff-program`
`ediff-diff3-program`
: These variables specify the programs to use to produce differences and do patching.

`ediff-diff-options`
`ediff-diff3-options`
: These variables specify the options to pass to the above utilities.

  In `ediff-diff-options`, it may be useful to specify options such as '`-w`' that ignore certain kinds of changes. However, Ediff does not let you use the option '`-c`', as it doesn't recognize this format yet.

`ediff-patch-program`
: The program to use to apply patches. Since there are certain incompatibilities between the different versions of the patch program, the best way to stay out of trouble is to use a GNU-compatible version. Otherwise, you may have to tune the values of the variables `ediff-patch-options`, `ediff-backup-specs`, and `ediff-backup-extension` as described below.

`ediff-patch-options`
: Options to pass to `ediff-patch-program`.

  Note: the '-b' and '-z' options should be specified in 'ediff-backup-specs', not in `ediff-patch-options`.

  It is recommended to pass the '-f' option to the patch program, so it won't ask questions. However, some implementations don't accept this option, in which case the default value of this variable should be changed.

`ediff-backup-extension`
: Backup extension used by the patch program. Must be specified, even if `ediff-backup-specs` is given.

`ediff-backup-specs`
: Backup directives to pass to the patch program. Ediff requires that the old version of the file (before applying the patch) is saved in a file named '`the-patch-file.extension`'. Usually 'extension' is '.orig', but this can be changed by the user, and may also be system-dependent. Therefore, Ediff needs to know the backup extension used by the patch program.

  Some versions of the patch program let the user specify '-b backup-extension'. Other versions only permit '-b', which (usually) assumes the extension '.orig'. Yet others force you to use '-z<backup-extension>'.

  Note that both 'ediff-backup-extension' and 'ediff-backup-specs' must be properly set. If your patch program takes the option '-b', but not '-b extension', the variable 'ediff-backup-extension' must still be set so Ediff will know which extension to use.

`ediff-custom-diff-program`
`ediff-custom-diff-options`
: Because Ediff limits the options you may want to pass to the `diff` program, it partially makes up for this drawback by letting you save the output from `diff` in your preferred format, which is specified via the above two variables.

The output generated by `ediff-custom-diff-program` (which doesn't even have to be a standard-style '`diff`'!) is not used by Ediff. It is provided exclusively so that you can refer to it later, send it over email, etc. For instance, after reviewing the differences, you may want to send context differences to a colleague. Since Ediff ignores the '`-c`' option in `ediff-diff-program`, you would have to run `diff -c` separately just to produce the list of differences. Fortunately, `ediff-custom-diff-program` and `ediff-custom-diff-options` eliminate this nuisance by keeping a copy of a difference list in the desired format in a buffer that can be displayed via the command *D*.

`ediff-patch-default-directory`
Specifies the default directory to look for patches.

Warning: Ediff does not support the output format of VMS `diff`. Instead, make sure you are using some implementation of POSIX `diff`, such as `gnudiff`.

7.9 Merging and diff3

Ediff supports three-way comparison via the functions `ediff-files3` and `ediff-buffers3`. The interface is the same as for two-way comparison. In three-way comparison and merging, Ediff reports if any two difference regions are identical. For instance, if the current region in buffer A is the same as the region in buffer C, then the mode line of buffer A will display '`[=diff(C)]`' and the mode line of buffer C will display '`[=diff(A)]`'.

Merging is done according to the following algorithm.

If a difference region in one of the buffers, say B, differs from the ancestor file while the region in the other buffer, A, doesn't, then the merge buffer, C, gets B's region. Similarly when buffer A's region differs from the ancestor and B's doesn't, A's region is used.

If both regions in buffers A and B differ from the ancestor file, Ediff chooses the region according to the value of the variable `ediff-default-variant`. If its value is `default-A` then A's region is chosen. If it is `default-B` then B's region is chosen. If it is `combined` then the region in buffer C will look like this:

```
#ifdef NEW  /* variant A */
difference region from buffer A
#else  /* variant B */
difference region from buffer B
#endif  /* NEW */
```

The actual strings that separate the regions copied from buffer A and B are controlled by the variable `ediff-combination-pattern`. Its value should be a list of three strings. The first is inserted before the difference region of buffer A; the second string goes between the regions; the third goes after region B, as shown in the above example.

In addition to the state of the difference, Ediff displays the state of the merge for each region. If a difference came from buffer A by default (because both regions A and B were different from the ancestor and `ediff-default-variant` was set to `default-A`) then '`[=diff(A) default-A]`' is displayed in the mode line. If the difference in buffer C came, say, from buffer B because the difference region in that buffer differs from the ancestor, but the region in buffer A does not (if merging with an ancestor) then '`[=diff(B) prefer-B]`' is displayed. The indicators default-A/B and prefer-A/B are inspired by Emerge and have the same meaning.

Another indicator of the state of merge is '`combined`'. It appears with any difference region in buffer C that was obtained by combining the difference regions in buffers A and B as explained above.

In addition to the state of merge and state of difference indicators, while merging with an ancestor file or buffer, Ediff informs the user when the current difference region in the (normally invisible) ancestor buffer is empty via the *AncestorEmpty* indicator. This helps determine if the changes made to the original in variants A and B represent pure insertion or deletion of text: if the mode line shows *AncestorEmpty* and the corresponding region in buffers A or B is not empty, this means that new text was inserted. If this indicator is not present and the difference regions in buffers A or B are non-empty, this means that text was modified. Otherwise, the original text was deleted.

Although the ancestor buffer is normally invisible, Ediff maintains difference regions there and advances the current difference region accordingly. All highlighting of difference regions is provided in the ancestor buffer, except for the fine differences. Therefore, if desired, the user can put the ancestor buffer in a separate frame and watch it there. However, on a TTY, only one frame can be visible at any given time, and Ediff doesn't support any single-frame window configuration where all buffers, including the ancestor buffer, would be visible. However, the ancestor buffer can be displayed by typing `/` to the control window. (Type `C-l` to hide it again.)

Note that the state-of-difference indicators '`=diff(A)`' and '`=diff(B)`' above are not redundant, even in the presence of a state-of-merge indicator. In fact, the two serve different purposes.

For instance, if the mode line displays '`=diff(B) prefer(B)`' and you copy a difference region from buffer A to buffer C then '`=diff(B)`' will change to '`diff-A`' and the mode line will display '`=diff(A) prefer-B`'. This indicates that the difference region in buffer C is identical to that in buffer A, but originally buffer C's region came from buffer B. This is useful to know because you can recover the original difference region in buffer C by typing `r`.

Ediff never changes the state-of-merge indicator, except in response to the `!` command (see below), in which case the indicator is lost. On the other hand, the state-of-difference indicator is changed automatically by the copying/recovery commands, `a`, `b`, `r`, `+`.

The `!` command loses the information about origins of the regions in the merge buffer (default-A, prefer-B, or combined). This is because recomputing differences in this case means running `diff3` on buffers A, B, and the merge buffer, not on the ancestor buffer. (It makes no sense to recompute differences using the ancestor file, since in the merging mode Ediff assumes that you have not edited buffers A and B, but that you may have edited buffer C, and these changes are to be preserved.) Since some difference regions may disappear as a result of editing buffer C and others may arise, there is generally no simple way to tell where the various regions in the merge buffer came from.

In three-way comparison, Ediff tries to disregard regions that consist entirely of white space. For instance, if, say, the current region in buffer A consists of the white space only (or if it is empty), Ediff will not take it into account for the purpose of computing fine differences. The result is that Ediff can provide a better visual information regarding the actual fine differences in the non-white regions in buffers B and C. Moreover, if the regions in buffers B and C differ in the white space only, then a message to this effect will be displayed.

In the merge mode, the share of the split between window C (the window displaying the merge-buffer) and the windows displaying buffers A and B is controlled by the variable `ediff-merge-window-share`. Its default value is 0.5. To make the merge-buffer window smaller, reduce this amount.

We don't recommend increasing the size of the merge-window to more than half the frame (i.e., to increase the value of `ediff-merge-window-share`) to more than 0.5, since it would be hard to see the contents of buffers A and B.

You can temporarily shrink the merge window to just one line by typing *s*. This change is temporary, until Ediff finds a reason to redraw the screen. Typing *s* again restores the original window size.

With a positive prefix argument, the *s* command will make the merge window slightly taller. This change is persistent. With '`-`' or with a negative prefix argument, the command *s* makes the merge window slightly shorter. This change also persistent.

Ediff lets you automatically ignore the regions where only one of the buffers A and B disagrees with the ancestor. To do this, set the variable `ediff-show-clashes-only` to non-`nil`.

You can toggle this feature interactively by typing *$*.

Note that this variable affects only the show next/previous difference commands. You can still jump directly to any difference region directly using the command *j* (with a prefix argument specifying the difference number).

The variable `ediff-autostore-merges` controls what happens to the merge buffer when Ediff quits. If the value is `nil`, nothing is done to the merge buffer—it will be the user's responsibility to save it. If the value is `t`, the user will be asked where to save the buffer and whether to delete it afterwards. It the value is neither `nil` nor `t`, the merge buffer is saved *only* if this merge session was invoked from a group of related Ediff session, such as those that result from `ediff-merge-directories`, `ediff-merge-directory-revisions`, etc. See Chapter 5 [Session Groups], page 14. This behavior is implemented in the function `ediff-maybe-save-and-delete-merge`, which is a hook in `ediff-quit-merge-hook`. The user can supply a different hook, if necessary.

The variable `ediff-autostore-merges` is buffer-local, so it can be set in a per-buffer manner. Therefore, use `setq-default` to globally change this variable.

7.10 Support for Version Control

Ediff supports version control and lets you compare versions of files visited by Emacs buffers via the function `ediff-revision`. This feature is controlled by the following variables:

`ediff-version-control-package`
: A symbol. The default is '`vc`'.

  If you are like most Emacs users, Ediff will use VC as the version control package. This is the standard Emacs interface to RCS, CVS, and SCCS.

However, if your needs are better served by other interfaces, you will have to tell Ediff which version control package you are using, e.g.,

```
(setq ediff-version-control-package 'rcs)
```

Apart from the standard 'vc.el', Ediff supports three other interfaces to version control: 'rcs.el', 'pcl-cvs.el', and 'generic-sc.el'. The package 'rcs.el' is written by Sebastian Kremer <sk@thp.Uni-Koeln.DE> and is available as

```
'ftp.cs.buffalo.edu:pub/Emacs/rcs.tar.Z'
'ftp.uni-koeln.de:/pub/gnu/emacs/rcs.tar.Z'
```

Ediff's interface to the above packages allows the user to compare the versions of the current buffer or to merge them (with or without an ancestor-version). These operations can also be performed on directories containing files under version control.

In case of 'pcl-cvs.el', Ediff can also be invoked via the function `run-ediff-from-cvs-buffer`—see the documentation string for this function.

7.11 Customizing the Mode Line

When Ediff is running, the mode line of 'Ediff Control Panel' buffer shows the current difference number and the total number of difference regions in the two files.

The mode line of the buffers being compared displays the type of the buffer ('A:', 'B:', or 'C:') and (usually) the file name. Ediff tries to be intelligent in choosing the mode line buffer identification. In particular, it works well with the 'uniquify.el' and 'mode-line.el' packages (which improve on the default way in which Emacs displays buffer identification). If you don't like the way Ediff changes the mode line, you can use `ediff-prepare-buffer-hook` to modify the mode line.

7.12 Miscellaneous

Here are a few other variables for customizing Ediff:

`ediff-split-window-function`

Controls the way you want the window be split between file-A and file-B (and file-C, if applicable). It defaults to the vertical split (`split-window-vertically`, but you can set it to `split-window-horizontally`, if you so wish. Ediff also lets you switch from vertical to horizontal split and back interactively.

Note that if Ediff detects that all the buffers it compares are displayed in separate frames, it assumes that the user wants them to be so displayed and stops splitting windows. Instead, it arranges for each buffer to be displayed in a separate frame. You can switch to the one-frame mode by hiding one of the buffers A/B/C.

You can also swap the windows where buffers are displayed by typing `~`.

`ediff-merge-split-window-function`

Controls how windows are split between buffers A and B in the merge mode. This variable is like `ediff-split-window-function`, but it defaults to `split-window-horizontally` instead of `split-window-vertically`.

`ediff-make-wide-display-function`
: The value is a function to be called to widen the frame for displaying the Ediff buffers. See the on-line documentation for `ediff-make-wide-display-function` for details. It is also recommended to look into the source of the default function `ediff-make-wide-display`.

  You can toggle wide/regular display by typing *m*. In the wide display mode, buffers A, B (and C, when applicable) are displayed in a single frame that is as wide as the entire workstation screen. This is useful when files are compared side-by-side. By default, the display is widened without changing its height.

`ediff-use-last-dir`
: Controls the way Ediff presents the default directory when it prompts the user for files to compare. If `nil`, Ediff uses the default directory of the current buffer when it prompts the user for file names. Otherwise, it will use the directories it had previously used for files A, B, or C, respectively.

`ediff-no-emacs-help-in-control-buffer`
: If `t`, makes *C-h* behave like the ⟨DEL⟩ key, i.e., it will move you back to the previous difference rather than invoking help. This is useful when, in an xterm window or a text-only terminal, the Backspace key is bound to *C-h* and is positioned more conveniently than the ⟨DEL⟩ key.

`ediff-toggle-read-only-function`
: This variable's value is a function that Ediff uses to toggle the read-only property in its buffers.

  The default function that Ediff uses simply toggles the read-only property, unless the file is under version control. For a checked-in file under version control, Ediff first tries to check the file out.

`ediff-make-buffers-readonly-at-startup nil`
: If t, all variant buffers are made read-only at Ediff startup.

`ediff-keep-variants`
: The default is `t`, meaning that the buffers being compared or merged will be preserved when Ediff quits. Setting this to `nil` causes Ediff to offer the user a chance to delete these buffers (if they are not modified). Supplying a prefix argument to the quit command (`q`) temporarily reverses the meaning of this variable. This is convenient when the user prefers one of the behaviors most of the time, but occasionally needs the other behavior.

  However, Ediff temporarily resets this variable to `t` if it is invoked via one of the "buffer" jobs, such as `ediff-buffers`. This is because it is all too easy to loose day's work otherwise. Besides, in a "buffer" job, the variant buffers have already been loaded prior to starting Ediff, so Ediff just preserves status quo here.

  Using `ediff-cleanup-hook`, one can make Ediff delete the variants unconditionally (e.g., by making `ediff-janitor` into one of these hooks).

`ediff-grab-mouse`
: Default is `t`. Normally, Ediff grabs mouse and puts it in its control frame. This is useful since the user can be sure that when he needs to type an Ediff

command the focus will be in an appropriate Ediff's frame. However, some users prefer to move the mouse by themselves. The above variable, if set to `maybe`, will prevent Ediff from grabbing the mouse in many situations, usually after commands that may take more time than usual. In other situation, Ediff will continue grabbing the mouse and putting it where it believes is appropriate. If the value is `nil`, then mouse is entirely user's responsibility. Try different settings and see which one is for you.

7.13 Notes on Heavy-duty Customization

Some users need to customize Ediff in rather sophisticated ways, which requires different defaults for different kinds of files (e.g., SGML, etc.). Ediff supports this kind of customization in several ways. First, most customization variables are buffer-local. Those that aren't are usually accessible from within Ediff Control Panel, so one can make them local to the panel by calling make-local-variable from within `ediff-startup-hook`.

Second, the function `ediff-setup` accepts an optional sixth argument which has the form `((`*var-name-1* `.` *val-1*`) (`*var-name-2* `.` *val-2*`) ...)`. The function `ediff-setup` sets the variables in the list to the respective values, locally in the Ediff control buffer. This is an easy way to throw in custom variables (which usually should be buffer-local) that can then be tested in various hooks.

Make sure the variable `ediff-job-name` and `ediff-word-mode` are set properly in this case, as some things in Ediff depend on this.

Finally, if you want custom-tailored help messages, you can set the variables `ediff-brief-help-message-function` and `ediff-long-help-message-function` to functions that return help strings.

When customizing Ediff, some other variables are useful, although they are not user-definable. They are local to the Ediff control buffer, so this buffer must be current when you access these variables. The control buffer is accessible via the variable `ediff-control-buffer`, which is also local to that buffer. It is usually used for checking if the current buffer is also the control buffer.

Other variables of interest are:

`ediff-buffer-A`
: The first of the data buffers being compared.

`ediff-buffer-B`
: The second of the data buffers being compared.

`ediff-buffer-C`
: In three-way comparisons, this is the third buffer being compared. In merging, this is the merge buffer. In two-way comparison, this variable is nil.

`ediff-window-A`
: The window displaying buffer A. If buffer A is not visible, this variable is nil or it may be a dead window.

`ediff-window-B`
: The window displaying buffer B.

`ediff-window-C`
: The window displaying buffer C, if any.

`ediff-control-frame`
: A dedicated frame displaying the control buffer, if it exists. It is non-nil only if Ediff uses the multiframe display, i.e., when the control buffer is in its own frame.

8 Credits

Ediff was written by Michael Kifer <kifer@cs.sunysb.edu>. It was inspired by emerge.el written by Dale R. Worley <drw@math.mit.edu>. An idea due to Boris Goldowsky <boris@cs.rochester.edu> made it possible to highlight fine differences in Ediff buffers. Alastair Burt <burt@dfki.uni-kl.de> ported Ediff to XEmacs, Eric Freudenthal <freudent@jan.ultra.nyu.edu> made it work with VC, Marc Paquette <marcpa@cam.org> wrote the toolbar support package for Ediff, and Hrvoje Niksic <hniksic@srce.hr> adapted it to the Emacs customization package.

Many people provided help with bug reports, patches, and advice. Without them, Ediff would not be nearly as useful as it is today. Here is a full list of contributors (I hope I didn't miss anyone):

```
Steve Baur (steve@xemacs.org),
Neal Becker (neal@ctd.comsat.com),
E. Jay Berkenbilt (ejb@ql.org),
Alastair Burt (burt@dfki.uni-kl.de),
Paul Bibilo (peb@delcam.co.uk),
Kevin Broadey (KevinB@bartley.demon.co.uk),
Harald Boegeholz (hwb@machnix.mathematik.uni-stuttgart.de),
Bradley A. Bosch (brad@lachman.com),
Michael D. Carney (carney@ltx-tr.com),
Jin S. Choi (jin@atype.com),
Scott Cummings (cummings@adc.com),
Albert Dvornik (bert@mit.edu),
Eric Eide (eeide@asylum.cs.utah.edu),
Paul Eggert (eggert@twinsun.com),
Kevin Esler (esler@ch.hp.com),
Robert Estes (estes@ece.ucdavis.edu),
Jay Finger (jayf@microsoft.com),
Xavier Fornari (xavier@europe.cma.fr),
Eric Freudenthal (freudent@jan.ultra.nyu.edu),
Job Ganzevoort (Job.Ganzevoort@cwi.nl),
Boris Goldowsky (boris@cs.rochester.edu),
Allan Gottlieb (gottlieb@allan.ultra.nyu.edu),
Thorbjoern Hansen (thorbjoern.hansen@mchp.siemens.de),
Xiaoli Huang (hxl@epic.com),
Lars Magne Ingebrigtsen (larsi@ifi.uio.no),
Larry Gouge (larry@itginc.com),
Karl Heuer (kwzh@gnu.org),
(irvine@lks.csi.com),
(jaffe@chipmunk.cita.utoronto.ca),
David Karr (dkarr@nmo.gtegsc.com),
Norbert Kiesel (norbert@i3.informatik.rwth-aachen.de),
Leigh L Klotz (klotz@adoc.xerox.com),
Fritz Knabe (Fritz.Knabe@ecrc.de),
Heinz Knutzen (hk@informatik.uni-kiel.d400.de),
Andrew Koenig (ark@research.att.com),
Ken Laprade (laprade@dw3f.ess.harris.com),
```

```
Will C Lauer (wcl@cadre.com),
Richard Levitte (levitte@e.kth.se),
Mike Long (mike.long@analog.com),
Martin Maechler (maechler@stat.math.ethz.ch),
Simon Marshall (simon@gnu.org),
Richard Mlynarik (mly@adoc.xerox.com),
Chris Murphy (murphycm@sun.aston.ac.uk),
Erik Naggum (erik@naggum.no),
Eyvind Ness (Eyvind.Ness@hrp.no),
Ray Nickson (nickson@cs.uq.oz.au),
David Petchey (petchey_david@jpmorgan.com),
Benjamin Pierce (benjamin.pierce@cl.cam.ac.uk),
Tibor Polgar (tlp00@spg.amdahl.com),
David Prince (dave0d@fegs.co.uk),
Paul Raines (raines@slac.stanford.edu),
Bill Richter (richter@math.nwu.edu),
C.S. Roberson (roberson@aur.alcatel.com),
Kevin Rodgers (kevin.rodgers@ihs.com),
Sandy Rutherford (sandy@ibm550.sissa.it),
Heribert Schuetz (schuetz@ecrc.de),
Andy Scott (ascott@pcocd2.intel.com),
Axel Seibert (axel@tumbolia.ppp.informatik.uni-muenchen.de),
Scott O. Sherman (Scott.Sherman@mci.com),
Richard Stallman (rms@gnu.org),
Richard Stanton (stanton@haas.berkeley.edu),
Ake Stenhoff (etxaksf@aom.ericsson.se),
Stig (stig@hackvan.com),
Peter Stout (Peter_Stout@cs.cmu.edu),
Chuck Thompson (cthomp@cs.uiuc.edu),
Ray Tomlinson (tomlinso@bbn.com),
Raymond Toy (toy@rtp.ericsson.se),
Jan Vroonhof (vroonhof@math.ethz.ch),
Philippe Waroquiers (philippe.waroquiers@eurocontrol.be),
Klaus Weber (gizmo@zork.north.de),
Ben Wing (wing@666.com),
Ilya Zakharevich (ilya@math.ohio-state.edu),
Eli Zaretskii (eliz@is.elta.co.il)
```

Index

F

G

M

P

R

U

V

Table of Contents