# Diff & Merge

| Operation | Keystroke | Function | Note |
|---|---|---|---|
| **Diff, Merge & Patch Files within Emacs**<br><br>See also: ∑ **Customize** | | Emacs has complete support to perform **text file diff**, 2-way merge and **3-way merge** operations. This nicely compares to **what's available outside Emacs**.<br>• Emacs has two built-in packages that support comparing files: **diff** and **ediff**.<br>  • ediff is more powerful than diff and more visual, using one buffer per file. ediff supports **3-way merge** and supports diff of directory trees.<br>• For **3-way merge** operations Emacs provides the built-in<br>  • ediff<br>  • smerge.<br>    • The smerge system is quite useful: if activated by **pel-use-smerge** set to auto, Emacs will automatically launch smerge when it detects a (D)VCS merge markup inside a file, similar to what Git or Mercurial will do when they cannot automatically complete a merge operation themselves.<br>      • smerge provides all the commands to perform the merge. If you prefer to use 3 buffers you can invoke ediff directly from smerge.<br>      • smerge inherit from the older but still supported **Emerge** built-in package.<br>      • PEL does not provide key bindings for Emerge as ediff commands are easier to use and more powerful.<br>PEL also provides support for the following extra external packages:<br>  📦 The **diffview-mode** package provides side-by-side diff format. 🔗 PEL activates it when **pel-use-diffview** is set to **t**.<br>  📦 The **ztree external package** provides a tree-view directory diff. 🔗 PEL activates it when **pel-use-ztree** is set to **t**. |
| **Open this PDF file.**<br>See also: ∑ **Help/Info** | `<f11> d <f1>` | **(pel-help-pdf** &optional OPEN-WEB-PAGE) | Open the ∑ **Diff & Merge** local PDF. If the prefix argument (like `C-u` or `M--`) is used, then it opens the remote GitHub hosted raw PDF instead. If the **pel-flip-help-pdf-arg** user-option is set it's the other way around. |
| | `<f12> <f1>` | | |
| **Customize PEL support for diff** | `<f11> d <f2>` | **(pel-customize-pel** &optional OTHER-WINDOW) | Customize PEL support for diff: smerge, ztree<br>• If OTHER-WINDOW is non-nil (use `C-u`), display in other window. |
| | `<f12> <f2>` | | |
| **Customize Emacs for specify diff mode**<br>• Adjusted for the current diff mode being used | `<f12> <f3>` | **(pel-customize-library** &optional OTHER-WINDOW) | Customize Emacs support for currently active diff mode.<br>For this & following: If OTHER-WINDOW non-nil (`C-u`), display in other window. |
| | `<f11> d <f3>` | | Customize Emacs support for diff, ediff, emerge, smerge, ztree. |
| | `<f11> d e <f3>` | | Customize Emacs ediff. |
| | `<f11> d s <f3>` | | Customize Emacs smerge |
| **Diff** | | The **diff-mode** provides a set of commands to compare files and buffers. This mode is used when the buffer holds a patch file created by various tools.<br>Use the following commands to compare files and buffers and enter the diff-mode. See the commands provided by the diff-mode below. |
| **Compare 2 files** | `<f11> d f` | **(diff** OLD NEW &optional SWITCHES NO-ASYNC) | Find and display the differences between OLD and NEW files.<br>• Prompt for NEW, then OLD files. |
| **Compare file with its backup** | `<f11> d k` | **(diff-backup** FILE &optional SWITCHES) | Diff this file with its backup file or vice versa.<br>• Uses the latest backup, if there are several numerical backups.<br>• If this file is a backup, diff it with its original.<br>• The backup file is the first file given to 'diff'.<br>• With prefix arg, prompt for diff switches. |
| **Compare buffer and associated file** | `<f11> d b` | **(diff-buffer-with-file** &optional BUFFER) | View the differences between BUFFER and its associated file. |
| **Compare current and other window** | `<f11> d w` | **(compare-windows** IGNORE-WHITESPACE) | Compare text in current window with text in another window. |
| | | • The **compare-windows-get-window-function** user option defines how to get another window. Set to compare-windows-get-recent-window by default.<br>• Compares the text starting at point in each window, moving over text in each one as far as they match.<br>• This command pushes the mark in each window at the prior location of point in that window.<br>• If both windows display the same buffer, the mark is pushed twice in that buffer: first in the other window, then in the selected window.<br>  ☝ Use this for simple comparison between 2 windows just to see if they have the same content.<br>• Split your frame in 2 windows, loaded with the buffer of the files you want to compare.<br>• Place point at the top of each buffer and issue the command.<br>• Point will be moved to the first difference in both window. If there is no difference point will be moved at the end of each window. |
| **diff-mode** commands | | Use the following commands in a buffer using the **diff-mode** major mode to manipulate the patch diff file and the **diff hunks**.<br>☝ The following commands are also available from the Diff menu on the menu-bar accessible via the `<f10>` key. See ∑ **Menus** |
| **diff-mode setup** | `<f12> <f4> ?` | **(pel-diff-show-status)** | Display diff-mode status info in mini-buffer: whether it jumps to new or old source. |
| **Toggle Next-Error-Follow minor mode** | `C-c C-f` | **(next-error-follow-minor-mode** &optional ARG) | Minor mode for compilation, occur and diff modes.<br>• With a prefix argument ARG, enable mode if ARG is positive, and disable it otherwise. If called from Lisp, enable mode if ARG is omitted or nil.<br>• When turned on, cursor motion in the compilation, grep, occur or diff buffer causes automatic display of the corresponding source code location. |
| **Jump to source file**<br>• open source of current hunk in a new window | `C-c C-c` | **(diff-goto-source** &optional OTHER-FILE EVENT) | Jump to the corresponding source line.<br>• 'diff-jump-to-old-file' (or its opposite if the OTHER-FILE prefix arg is given) determines whether to jump to the **old** or the **new** file.<br>• If the prefix arg is bigger than 8 (for example with `C-u C-u` then 'diff-jump-to-old-file' is also set, for the next invocations. Use `<f12> <f4> ?` to see status. |
| **To next hunk** | • `M-n`<br>• `C-M-i` | **(diff-hunk-next** &optional COUNT) | Go to the next COUNT'th hunk. |
| **To previous hunk** | • `M-p`<br>• `<Esc> <backtab>` | **(diff-hunk-prev** &optional COUNT) | Go to the previous COUNT'th hunk |
| **To next file** | • `M-}`<br>• `M-N`<br>• `<f6> <down>` | **(diff-file-next** &optional COUNT) | Go to the next COUNT'th file. |
| **To previous file** | • `M-{`<br>• `M-P`<br>• `<f6> <up>` | **(diff-file-prev** &optional COUNT) | Go to the previous COUNT'th file |
| **Show all files present in diff inside an occur buffer** | `<f6> o` | **(pel-diff-hunk-files-occur** &optional NLINES) | Show hunk files of current path patch inside an occur buffer.<br>• Each line shown with NLINES before & after, or -NLINES before if NLINES < 0.<br>• NLINES defaults to 0, overriding list-matching-lines-default-context-lines.<br>• If a region is defined the search is restricted to the region. See **occur search**. |
| **Restrict view to current hunk or file**<br>See ∑ **Narrowing** | `C-c C-n` | **(diff-restrict-view** &optional ARG) | Restrict the view to the current hunk.<br>• If the prefix ARG is given, restrict the view to the current file instead.<br>• This is a buffer narrowing type of command.<br>• Use `C-x n w` to widen the buffer back. |
| **Apply current hunk to source file** | `C-c C-a` | **(diff-apply-hunk** &optional REVERSE) | Apply current **hunk** (*not all* hunks for the file!) to the source file and go to the next.<br>• By default, the new source file is patched, but if the variable 'diff-jump-to-old-file' is non-nil, then the old source file is patched instead (some commands, such as 'diff-goto-source' can change the value of this variable when given an appropriate prefix argument). With a prefix argument, REVERSE the hunk. |
| **Apply diff with Ediff** | `C-c C-e` | **(diff-ediff-patch)** | Call 'ediff-patch-file' on the current buffer:<br>  Query for a file name, and then run Ediff by patching that file.<br>  • If optional PATCH-BUF is given, use the patch in that buffer & don't ask user.<br>  • If prefix argument ARG, then: if even argument, assume that the patch is in a buffer. If odd -- assume it is in a file. |

| Operation | Keystroke | Function | Note |
|---|---|---|---|
| **Highlight changes in hunk** | `C-c C-b` | **(diff-refine-hunk)** | Highlight changes of hunk at point at a finer granularity. |
| **Test if hunk can be applied** | `C-c C-t` | **(diff-test-hunk** &optional REVERSE) | See whether it's possible to apply the current hunk.<br>With a prefix argument, try to REVERSE the hunk. |
| **Convert unified diff to context diff** | `C-c C-d` | **(diff-unified->context** START END) | Convert unified diffs to context diffs.<br>• START and END are either taken from the region (if a prefix arg is given) or else cover the whole buffer. |
| **Reverse direction of diff** | `C-c C-r` | **(diff-reverse-direction** START END) | Reverse the direction of the diffs.<br>• START and END are either taken from the region (if a prefix arg is given) or else cover the whole buffer. |
| **Split hunk** | `C-c C-s` | **(diff-split-hunk)** | Split the current (unified diff) hunk at point into two hunks. |
| **Convert context diff to unified diff** | `C-c C-u` | **(diff-context->unified** START END &optional TO-CONTEXT) | Convert context diffs to unified diffs.<br>• START and END are either taken from the region (when it is highlighted) or else cover the whole buffer.<br>• With a prefix argument, convert unified format to context format. |
| **Re-diff current hunk, ignore whitespace** | `C-c C-w` | **(diff-ignore-whitespace-hunk)** | Re-diff the current hunk, ignoring whitespace differences. |
| **Re-diff all hunks, ignore whitespace** | `<f6> w` | **(pel-diff-ignore-whitespace-in-hunks)** | Re-diff all hunks in buffer, ignoring whitespace differences. |
| **Ediff**<br><br>See also: ∑ Scrolling | Ediff sessions have several commands, shown in the Ediff Quick Help buffer.<br> • Type **?** to toggle Ediff Quick Help from 1 line to multiple that shows all available commands.<br> • For more info about a command, place point on the command character and type **RET**: Emacs will open the **Ediff Quick Help Commands** Info buffer.<br> ☝ While showing 2 or 3 buffer/files in windows Ediff provides the **v/V** keys for scrolling up/down.<br> PEL scroll sync commands can also be used to provide single line scroll synced between the windows. See the `<f11> \|` commands in table. | | |
| **Display Ediff Manual** | `<f11> d e ?` | **(ediff-documentation** &optional NODE) | Display Ediff's manual.<br>• With optional NODE, goes to that node. |
| **Ediff 2 files**<br><br>★★ | `<f11> d 2` | **(pel-ediff-2files** &optional N) | Run ediff-files on the files of current and the other window.<br>• Select the current file and the other file without prompting.<br>• With numeric argument if N is in [2,8] range, select other window identified by the direction corresponding to the cursor in a numeric keypad:<br> 8 := 'up<br> 4 := 'left  5 := 'current  6 := 'right<br> 2 := 'down |
| **Display registry of active Ediff sessions** | `<f11> d e R` | • **(eregistry)**<br>• **(ediff-show-registry)** | Display registry of all active Ediff sessions. |
| **Ediff file against previous revision** | `<f11> d r` | **(pel-ediff-revisions)** | Run ediff-revision on the file in the current window.<br>• Prompts for revisions, default to current copy and last commit. |
| **Compare buffer with its file on disk** | `<f11> d e b f` | **(ediff-current-file)** | Compare the buffer with its file on disk. This function can be used as a safe version of revert-buffer. |
| **Compare 2 buffers** | `<f11> d e b b` | **(ediff-buffers** BUFFER-A BUFFER-B &optional STARTUP-HOOKS JOB-NAME) | Compare 2 buffers.<br>• Prompts for buffer A and buffer B |
| **Compare 3 buffers** | `<f11> d e b 3` | **(ediff-buffers3** BUFFER-A BUFFER-B BUFFER-C &optional STARTUP-HOOKS JOB-NAME) | Compare 3 buffers.<br>• Prompts for buffer A, buffer B and buffer C. |
| **Compare file with its backup**<br>∑ Autosave/backup | `<f11> d e f k` | **(ediff-backup** FILE) | Compare a file with its backup. If there are several numerical backups, use the latest.<br>• If the file is itself a backup, then compare it with its original. |
| **Compare 2 files** | `<f11> d e f f` | • **(ediff** FILE-A FILE-B &optional STARTUP-HOOKS)<br>• **(ediff-files** FILE-A FILE-B &optional STARTUP-HOOKS) | Compare 2 files. Uses either diff or ediff-files.<br>• Prompts for file A and B.<br>• PEL provide a shortcut function **pel-ediff-2files** mapped to `<f11> d 2` which does not prompt. See above. |
| **Compare 3 files** | `<f11> d e f 3` | • **(ediff3** FILE-A FILE-B FILE-C &optional STARTUP-HOOKS)<br>• **(ediff-files3** FILE-A FILE-B FILE-C &optional STARTUP-HOOKS) | Compare 3 files.<br>• Prompts for file A, B and C. |
| **Compare revision of buffer with file revision** | `<f11> d e f r` | **(ediff-revision** &optional FILE STARTUP-HOOKS) | Compare versions of the current buffer, if the buffer is visiting a file under VCS.<br>• Prompts for the file name and each of its revisions.<br>• PEL provide a shortcut function **pel-ediff-revision** mapped to `<f11> d r`. |
| **Compare versions of files in a given directory** | `<f11> d e d r` | • **(edir-revisions** DIR1 REGEXP)<br>• **(ediff-directory-revisions** DIR1 REGEXP) | Compare versions of files in a given directory.<br>• Ediff selects only the files that are under version control.<br>• Prompts for directory and regexp to identify files: if empty: selects all files. |
| **Compare text in 2 windows word-by-word** | `<f11> d e w w` | **(ediff-windows-wordwise** DUMB-MODE &optional WIND-A WIND-B STARTUP-HOOKS) | Compare text visible in 2 windows word-by-word.<br>• Uses current and other (next) window. |
| **Compare text in 2 windows line-by-line** | `<f11> d e w l` | **(ediff-windows-linewise** DUMB-MODE &optional WIND-A WIND-B STARTUP-HOOKS) | Compare text visible in 2 windows line-by-line.<br>• Uses current and other (next) window. |
| **Compare 2 regions word-by-word** | `<f11> d e r w` | **(ediff-regions-wordwise** BUFFER-A BUFFER-B &optional STARTUP-HOOKS) | Compare text visible in 2 regions word-by-word.<br>• Prompts for the 2 buffers and regions. |
| **Compare 2 regions line-by-line** | `<f11> d e r l` | **(ediff-regions-linewise** BUFFER-A BUFFER-B &optional STARTUP-HOOKS) | Compare text visible in 2 regions line-by-line.<br>• Prompts for the 2 buffers and regions. |
| **Side-by-Side Diff**<br><br>During that mode: | Using 📦 diffview-mode package ☑ PEL activates it when **pel-use-diffview-mode** is set to **t**.<br><br>• **}** : Next file<br>• **{** : Previous file<br>• **l** : Align windows<br>• **q** : Quit | • With PEL, use **pel-toggle-scroll-sync**, mapped to `<f11> \|\|` to scroll both windows.<br>  • See ∑ Scrolling | • To return to the original window layout you can use **winner-undo** `<f11> w p`, with PEL.<br>  • See ∑ Windows *(end of page 4)* |
| current buffer | `<f11> d \|` | **(diffview-current)** | Show current diff buffer in a side-by-side view. |
| current region | `<f11> d M-\|` | **(diffview-region)** | Show current diff region in a side-by-side view. |
| **Compare Directories** | The built-in Ediff can compare 2 or 3 directories.<br>📦 The **ztree external package** provides a tree-view directory diff. ☑ PEL activates it when **pel-use-ztree** is set to **t**. | | |
| **Compare common files in 2 directories** | `<f11> d e d d` | • **(edirs** DIR1 DIR2 REGEXP)<br>• **(ediff-directories** DIR1 DIR2 REGEXP) | Compare files common to two directories.<br>• Prompts for directory A & B and a regexp to identify files. If empty: select all files. |
| **Compare common files in 3 directories** | `<f11> d e d 3` | • **(edirs3** DIR1 DIR2 DIR3 REGEXP)<br>• **(ediff-directories3** DIR1 DIR2 DIR3 REGEXP) | Compare files common to three directories.<br>• Prompts for directory A, B and C and a regexp to identify files: if empty: selects all files. |

| Operation | Keystroke | Function | Note |
|---|---|---|---|
| **Compare 2 directories with ztree-diff** | `<f11> d z` | **(ztree-diff** DIR1 DIR2) | Open an interactive buffer with the directory tree of the path given, and highlight of file differences between the directories. DIR1 := left directory. DIR2:=right directory<br>• Interactively: prompts for the 2 directories.<br>📦 Requires the ztree external package. 🔲 PEL activates it if **pel-use-ztree** is **t**. |
| Ztree Diff buffer keys:<br><br>• **Ediff 2 files** ☛ | • `<SPACE>` : Open/close directory<br>• `<TAB>` : switch between panels<br>• `<RET>` : Open/close directory \| **Ediff 2 files** /open orphan file<br>• `x` : Toggle expand/collapse of all nodes of the subtree.<br>⚠️ Use **x** with care! On large directory trees it may take a long time. I have see Emacs hang when typing **x** again during that time.<br>• `<f5>` : force full rescan, re-write buffer (useful when changing window size). | | • `C` : copy current file/directory to the directory shown in the other side (prompts)<br>• `D` : delete current file/directory (prompts)<br>• `h` : toggle display of identical files/directories.<br>• `H` : toggle display of filtered files.<br>• `r` : rescan/refresh current file/directory<br>• `v` : visit current file in view-mode . In view-mode the following keys are available (see ∑ **Buffers** for a complete list):<br>    • q: quit view-mode and return to Ztree Diff<br>    • e: leave view-mode, edit/visit the file normally. |
| **Ediff Merge** | Use the following commands to perform **3-way merges** within Emacs using the merge capability of the ediff built-in package. | | |
| **Merge 2 files** | `<f11> d e m f` | • **(ediff-merge** FILE-A FILE-B &optional STARTUP-HOOKS MERGE-BUFFER-FILE)<br>• **(ediff-merge-files** FILE-A FILE-B &optional STARTUP-HOOKS MERGE-BUFFER-FILE) | Merge two files without ancestor.<br>• Prompt for FILE-A and FILE-B, the names of the files to be merged.<br>• The result is stored into an *ediff-merge* buffer, not a file.<br>  • Save it into a file with `C-x C-s`. See **screenshot example in PEL manual**. |
| **Merge 2 files with ancestor** | `<f11> d e m F` | • **(ediff-merge-with-ancestor** FILE-A FILE-B FILE-ANCESTOR &optional STARTUP-HOOKS MERGE-BUFFER-FILE)<br>• (ediff-merge-files-with-ancestor FILE-A FILE-B FILE-ANCESTOR &optional STARTUP-HOOKS MERGE-BUFFER-FILE) | Merge two files with ancestor.<br>• Prompt for FILE-A and FILE-B, the names of the files to be merged, and FILE-ANCESTOR, the name of the ancestor file.<br>• The result is stored into an *ediff-merge* buffer, not a file.<br>  • Save it into a file with `C-x C-s`. See **screenshot example in PEL manual**. |
| **Merge 2 buffers** | `<f11> d e m b` | **(ediff-merge-buffers** BUFFER-A BUFFER-B &optional STARTUP-HOOKS JOB-NAME MERGE-BUFFER-FILE) | Merge buffers without ancestor.<br>• Prompt for BUFFER-A and BUFFER-B, the buffers to be merged. |
| **Merge 2 buffers with ancestor** | `<f11> d e m B` | **(ediff-merge-buffers-with-ancestor** BUFFER-A BUFFER-B BUFFER-ANCESTOR &optional STARTUP-HOOKS JOB-NAME MERGE-BUFFER-FILE) | Merge buffers with ancestor.<br>• Prompts for BUFFER-A and BUFFER-B, the buffers to be merged, and BUFFER-ANCESTOR, their ancestor. |
| **Merge versions of files in a directory** | `<f11> d e m d` | • **(edir-merge-revisions** DIR1 REGEXP &optional MERGE-AUTOSTORE-DIR)<br>• **(ediff-merge-directory-revisions** DIR1 REGEXP &optional MERGE-AUTOSTORE-DIR) | Merge versions of files in a given directory.<br>• Ediff selects only the files that are under version control.<br>• Prompts for directory and regexp to identify files: if empty: selects all files. |
| **Merge versions of files in a directory using other versions as their ancestors** | `<f11> d e m D` | • **(edir-merge-revisions-with-ancestor** DIR1 REGEXP &optional MERGE-AUTOSTORE-DIR)<br>• **(ediff-merge-directory-revisions-with-ancestor** DIR1 REGEXP &optional<br>• MERGE-AUTOSTORE-DIR) | Merge versions of files in a given directory using other versions as ancestors.<br>• Ediff selects only the files that are under version control.<br>• Prompts for directory and regexp to identify files: if empty: selects all files. |
| **Merge file commons to 2 directories** | `<f11> d e m c` | • **(edirs-merge** DIR1 DIR2 REGEXP &optional MERGE-AUTOSTORE-DIR)<br>• **(ediff-merge-directories** DIR1 DIR2 REGEXP &optional MERGE-AUTOSTORE-DIR) | Merge files common to two directories.<br>• Run Ediff on a pair of directories, DIR1 and DIR2, merging files that have the same name in both.<br>• The third argument, REGEXP, is nil or a regular expression; only file names that match the regexp are considered.<br>• MERGE-AUTOSTORE-DIR is the directory in which to store merged files. |
| **Merge file commons to 2 directories with ancestors** | `<f11> d e m C` | • **(edirs-merge-with-ancestor** DIR1 DIR2 ANCESTOR-DIR REGEXP &optional MERGE-AUTOSTORE-DIR)<br>• **(ediff-merge-directories-with-ancestor** DIR1 DIR2 ANCESTOR-DIR REGEXP &optional MERGE-AUTOSTORE-DIR) | Merge files in directories DIR1 and DIR2 using files in ANCESTOR-DIR as ancestors.<br>• Ediff merges files that have identical names in DIR1, DIR2.<br>• If a pair of files in DIR1 and DIR2 doesn't have an ancestor in ANCESTOR-DIR, Ediff will merge without ancestor.<br>• The fourth argument, REGEXP, is nil or a regular expression; only file names that match the regexp are considered. |
| **Merge 2 versions of visited file** | `<f11> d e m r` | **(ediff-merge-revisions** &optional FILE STARTUP-HOOKS MERGE-BUFFER-FILE) | Merge two versions of the file visited by the current buffer. |
| **Merge 2 versions of visited file with ancestor** | `<f11> d e m R` | **(ediff-merge-revisions-with-ancestor** &optional FILE STARTUP-HOOKS MERGE-BUFFER-FILE) | Merge two versions of the file visited by the current buffer with ancestor. |
| **Specialized Ediff** | Some packages provide specialization of Ediff-based comparisons. | | |
| **ParInfer EDiff**<br>**Diff current code before/.after ParInfer modifications**<br>∑🌐 - Emacs Lisp | • `<f12> a D`<br>• `<f11> SPC l a D` | **(parinfer-diff)** | Diff current code and the code after applying Indent Mode in Ediff.<br>Use this to browse and apply the changes.<br>☛📦 Requires the parinfer package. ⚠️ This is an obsolete package.<br>☛🔲 PEL activates this when the **pel-use-parinfer** user option is set to **t**. |
| **Smerge** | Use the built-in smerge package to edit files that contain 3-way merge conflict annotations placed by a 3-way merge operation that requires your input.<br>Start the merge session by executing the smerge-start-session or have it start<br>• If you want to automatically launch a smerge session on files that contain diff conflict annotations, set **pel-use-smerge** to auto.<br>  • Conflict annotation string is a string like "<<<<<<<" that starts at the beginning of a line.<br>When a merge session is active:<br>  • the merge menu is available. See ∑ **Menus**<br>  • with PEL, the `<f6> s` key acts as a prefix to the smerge commands. | | |
| **Start a smerge session** | `<f11> d s` | **(smerge-start-session)** | Turn on 'smerge-mode' and move point to first conflict marker.<br>If no conflict maker is found, turn off 'smerge-mode'. |
| **Move to next conflict** | • `C-c ^ n`<br>• `<f6> s n` | **(smerge-next** &optional COUNT) | Go to the next COUNT'th conflict. |
| **Move to previous conflict** | • `C-c ^ p`<br>• `<f6> s p` | **(smerge-prev** &optional COUNT) | Go to the previous COUNT'th conflict |
| **Keep all** | • `C-c ^ a`<br>• `<f6> s a` | **(smerge-keep-all)** | Concatenate all versions. |
| **Revert to base** | • `C-c ^ b`<br>• `<f6> s b` | **(smerge-keep-base)** | Revert to the base version. |
| **Keep current** | • `C-c ^ RET`<br>• `<f6> s RET` | **(smerge-keep-current)** | Use the current (under the cursor) version. |

| Operation | Keystroke | Function | Note |
|---|---|---|---|
| **Keep lower** | • `C-c ^ l`<br>• `<f6> s l` | **(smerge-keep-lower)** | Keep the "lower" version of a merge conflict.<br>  In a conflict that looks like:<br>    `<<<<<<<`<br>    `UUU`<br>    `=======`<br>    `LLL`<br>    `>>>>>>>`<br>  this keeps "LLL". |
| **Keep upper** | • `C-c ^ u`<br>• `<f6> s u`<br>• `C-c ^ m` | **(smerge-keep-upper)** | Keep the "upper" version of a merge conflict.<br>  In a conflict that looks like:<br>    `<<<<<<<`<br>    `UUU`<br>    `=======`<br>    `LLL`<br>    `>>>>>>>`<br>  this keeps "UUU". |
| **Auto-combine** | `<f6> s M-c` | **(smerge-auto-combine)** | Automatically combine conflicts that are near each other. |
| **Combine with next** | • `C-c ^ C`<br>• `<f6> s C` | **(smerge-combine-with-next)** | Combine the current conflict with the next one. |
| **Diff base & lower** | • `C-c ^ = >`<br>• `<f6> s >` | **(smerge-diff-base-lower)** | Diff 'base' and 'lower' version in current conflict region. |
| **Diff base & upper** | • `C-c ^ = <`<br>• `<f6> s <` | **(smerge-diff-base-upper)** | Diff 'base' and 'upper' version in current conflict region. |
| **Diff upper & lower** | • `C-c ^ = =`<br>• `<f6> s =` | **(smerge-diff-upper-lower)** | Diff 'upper' and 'lower' version in current conflict region. |
| **Invoke ediff** | • `C-c ^ E`<br>• `<f6> s e` | **(smerge-ediff** &optional NAME-UPPER NAME-LOWER NAME-BASE) | Invoke ediff to resolve the conflicts.<br>NAME-UPPER, NAME-LOWER, and NAME-BASE, if non-nil, are used for the buffer names. |
| **Remove current** | `<f6> s M-k` | **(smerge-kill-current)** | Remove the current (under the cursor) version. |
| **Insert diff3 conflict markers** | `<f6> s M-C` | **(smerge-makeup-conflict** PT1 PT2 PT3 &optional PT4) | Insert diff3 markers to make a new conflict.<br>• Uses point and mark for two of the relevant positions and previous marks for the other ones.<br>• By default, makes up a 2-way conflict, with a C-u prefix, makes up a 3-way conflict. |
| **Refine highlight** | • `C-c ^ R`<br>• `<f6> s R` | **(smerge-refine** &optional PART) | Highlight the words of the conflict that are different.<br>• For 3-way conflicts, highlights only two of the three parts.<br>• A numeric argument PART can be used to specify which two parts;<br>• repeating the command will highlight other two parts. |
| **Resolve conflict at point** | • `C-c ^ r`<br>• `<f6> s r` | **(smerge-resolve** &optional SAFE) | Resolve the conflict at point intelligently.<br>• This relies on mode-specific knowledge and thus only works in some major modes.<br>• Uses 'smerge-resolve-function' to do the actual work. |
| **Resolve all conflicts** | `<f6> s M-r` | **(smerge-resolve-all)** | Perform automatic resolution on all conflicts. |
| **Swap upper and lower** | `<f6> s M-s` | **(smerge-swap)** | Swap the "Upper" and the "Lower" chunks.<br>• Can be used before things like 'smerge-keep-all' or 'smerge-resolve' where the ordering can have some subtle influence on the result, such as preferring the spacing of the "Lower" chunk. |
| **Patch files** | Emacs supports **merging a patch** into a local file or a buffer.<br>• Patch files can be created by Emacs using diff and other tools.  Patch are normally using the **unified diff format**.  See the above sections for commands that can create patch files.<br>• The Emacs manual section on **Sending Patches on GNU Emacs** provides the following useful bit of information:<br>  • Use `diff -u -F'^[_a-zA-Z0-9$]\+ *('` when making diffs of C code. This shows the name of the function that each change occurs in.<br>  • Use `diff -u` to make your diffs. Diffs without context are hard to install reliably.<br>• The Emacs commands you can use to apply a patch to a file are described below. | | |
| **Patch file(s) and compare** | `<f11> d e p f`<br>`<f11> d p f` | • **(epatch** &optional ARG PATCH-BUF)<br>• **(ediff-patch-file** &optional ARG PATCH-BUF) | Query for a file name, and then run Ediff by patching that file.<br>• If optional PATCH-BUF is given, use the patch in that buffer and don't ask the user.<br>• If prefix argument ARG, then: if even argument, assume that the patch is in a buffer.  If odd -- assume it is in a file.<br>If a hunk cannot be applied, describe the problem inside the *ediff-message* buffer.<br>• It also creates a reject file that describes the hunk that could not be applied.  The file is stored inside the same directory as the target file, uses the same file name with '.rej' appended to the file name. This may happen if a hunk describes an addition that is already present inside the target file. |
| **Patch a buffer then compare** | `<f11> d e p b`<br>`<f11> d p b` | • **(epatch-buffer** &optional ARG PATCH-BUF)<br>• **(ediff-patch-buffer** &optional ARG PATCH-BUF) | Run Ediff by patching the buffer specified at prompt.<br>• Without the optional prefix ARG, asks if the patch is in some buffer and prompts for the buffer or a file, depending on the answer.<br>• With ARG=1, assumes the patch is in a file and prompts for the file.<br>• With ARG=2, assumes the patch is in a buffer and prompts for the buffer.<br>• PATCH-BUF is an optional argument, which specifies the buffer that contains the patch.  If not given, the user is prompted according to the prefix argument. |

## DIff & Merge — References

| | |
|---|---|
| **emacs vdiff github project** | |
| **Using Emacs as a merge program @ Mercurial Wiki** | |
| **GNU Emacs Ediff Manual** | Describes built-in Ediff:  file diff of 2 or 3 files, 3-way merge, diff with backup, directory level diffs, version control aware... |
| **vdiff.el** | Github home of vdiff.el, a package that implements a diff tool similar to vimdiff. |
| | |
| **Diff Patch** | • **patch @ Wikipedia** provides a good and concise introduction to the concept of diff and patch |
| **Handling Patch Rejects** | |
| | |