Sega Dreamcast

Dreamcast GNUPro™ Toolkit Advanced Topics

Rebuilding from Source

GNU Online Documentation

Reporting Problems

Legal Notices

Comparing & Merging Differences

Important InformMation

This documentation has been provided courtesy of CYGNUS. The contents are applicable to GNUPro™ Toolkit development, however, all references to development support offered by CYGNUS should be ignored.

Technical support for this product as it applies to the Sega Dreamcast™ development environment should be directed to Sega Third Party Developer Technical Support at 415/701-4060. Future updates and/or additional information may also be found at Sega's DTS Website at, **`http//:www.dts.sega.com/NextGen`**

Copyright © 1988-1998 Cygnus

Permission is granted to make and distribute verbatim copies of this documentation provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the "GNU General Public License" is included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

This documentation has been prepared by Cygnus.

All rights reserved.

GNUPro™, the GNUPro™ logo, and the Cygnus logo are trademarks of Cygnus. All other brand and product names are trademarks of their respective owners.

To contact the Cygnus Technical Publications staff, email: doc@cygnus.com.

Part #: 300-400-10100046

GNUPro Warranty

The GNUPro Toolkit is free software, covered by the GNU General Public License, and you are welcome to change it and/or distribute copies of it under certain conditions. This version of GNUPro Toolkit is supported for customers of Cygnus.

For non-customers, GNUPro Toolkit software has NO WARRANTY.

Because this software is licensed free of charge, there are no warranties for it, to the extent permitted by applicable law. Except when otherwise stated in writing, the copyright holders and/or other parties provide the software “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the software is with you. Should the software prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event, unless required by applicable law or agreed to in writing, will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

How to Contact Cygnus

Use the following means to contact Cygnus.

Cygnus Headquarters
1325 Chesapeake Terrace
Sunnyvale, CA 94089 USA
Telephone (toll free): +1 800 CYGNUS-1
Telephone (main line): +1 408 542 9600
Telephone (hotline): +1 408 542 9601
FAX: +1-408 542 9699
(Faxes are answered 8 a.m.–5 p.m., Monday through Friday.)
email: `info@cygnus.com`
Website: `www.cygnus.com`.

Cygnus United Kingdom
36 Cambridge Place
Cambridge CB2 1NS
United Kingdom
Telephone: +44 1223 728728
FAX: +44 1223 728728
email: `info@cygnus.co.uk/`

Cygnus Japan
Nihon Cygnus Solutions
Madre Matsuda Building
4-13 Kioi-cho Chiyoda-ku
Tokyo 102-0094
Telephone: +81 3 3234 3896
FAX: +81 3 3239 3300
email: `info@cygnus.co.jp`
Website: `http://www.cygnus.co.jp/`

Use the hotline (+1 408 542 9601) to get help, although the most reliable way to resolve problems with GNUPro Toolkit is by using email: `bugs@cygnus.com`.

Contents

GNU Online Documentation

Reporting Problems

Comparing & Merging Differences

GNPRO TOOLKIT™

Rebuilding from Source

98r1
July, 1998

CYGNUS

Copyright © 1988-1998 Cygnus

Permission is granted to make and distribute verbatim copies of this documentation provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the "GNU General Public License" are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

This documentation has been prepared by Cygnus.

All rights reserved.

GNUPro™, the GNUPro™ logo, and the Cygnus logo are trademarks of Cygnus. All other brand and product names are trademarks of their respective owners.

To contact the Cygnus Technical Publications staff, email: `doc@cygnus.com`.

1

Configuration

The following documentation explains how to configure your system before you rebuild the binaries from source code for a given toolchain.

You'll need to provide the following specifications when rebuilding:

- Where to install the binaries; see "Configuring the install location" on page 4.
- Which target system will run your final target machine's code; see "Configuring the target" on page 5.
- Preparing sources so that you get what you expect; see "Running configure" on page 6

Configuring the install location

The `--prefix=` option specifies the base installation path for the entire toolkit. In the following descriptions, the name of your current GNUPro software release is written in italics as *`release_name`*.

- To install the GNUPro Toolkit in **`/opt/cygnus/`***`release_name`*, use the following example's directive.

  `--prefix=/opt/cygnus/`*`release_name`*

- To install the GNUPro Toolkit in **`/usr/cygnus/`***`release_name`*, use the following example's directive.

  `--prefix=/usr/cygnus/`*`release_name`*

GNUPro binaries are typically located in `/usr/cygnus/`*`release_name`*.

To rebuild the GNUPro Toolkit for a Windows system and install it in the same location as the Cygnus binaries, use the following input.

```
--prefix=/usr/cygnus/release_name
–-exec-prefix=/usr/cygnus/release_name/H-sparc-sun-solaris2
```

`--exec-prefix=` is useful for sites with multiple host architectures in a networked, shared file system.

WARNING! The values of both `--prefix=` and `--exec-prefix=` must be ***absolute pathnames***.

The correct `--exec-prefix=` value for your release uses the following example's usage (where *`release_name`* is the current version's name).

`/usr/cygnus/`*`release_name`*`/H-hostspec`.

See also "Multiple host builds" on page 9.

Configuring the target

If you are targeting an embedded system, remember at configure time to specify the option, `--target=`. The `--target=` value is printed on the CD-ROM you receive from Cygnus. It will be a hyphenated string indicating *target* and *output format*.

For example, `m68k-elf` or `powerpc-eabi` would be two such target names with their appropriate file formats.

NOTE: To build a native toolkit, do ***not*** use `--target=`. For instance, programs such as the GNU debugger or the GNU compiler will run on an IRIX 5 system and output programs which also run on an IRIX 5 system. In this situation, the `--target=` option is unnecessary. `configure` will default to native configuration when using `--target`.

WARNING! For VxWorks users building a cross toolkit with the target board running the VxWorks from Wind River Systems, remember to add the `--with-includes=` command option to `configure`, designating the value for header files with your VxWorks distribution.

If the VxWorks tools are in `/opt/wrs/vxworks-5.2/`, then the header files are probably in `/opt/wrs/vxworks-5.2/h`.

Add the `--with-includes=/opt/wrs/vxworks-5.2/h` command option when you run `configure`. If you do not add this command line option, your toolchain will not work.

Running configure

The first and most important step in preparing source code to run on your system is to *configure* it so that, when built, the program exhibits the behavior you expect. The configuration process automatically prepares a Makefile containing default and/or customized information for your site and for your hardware/software system. If you are building for more than one platform, you must configure, compile, and install on each platform.

Source code is normally in `/usr/cygnus/release_name/src`. Good practice is to configure and build the source in another directory; any other clean directory will do. The build process may take hundreds of megabytes of disk space.

WARNING! ***Never build your toolkit in the source directory.***

The default action for `configure` is to configure a native toolchain for the host on which you run the script. At minimum, specify `--prefix=` to point to your installation directory. For cross-compiler toolchains, you must also specify `--target`. The following examples use a directory for the build location with the following name.

`/usr/cygnus/release_name/build.`

To configure a toolchain in `/opt/cygnus` targeting an Hitachi SH embedded board, use the following example's steps.

```
% mkdir /usr/cygnus/release_name/build
% cd /usr/cygnus/release_name/build
% /usr/cygnus/release_name/src/configure --target=sh-hms\
% --prefix=/opt/cygnus/release_name
```

To configure a native toolchain, use the following example.

```
% mkdir /usr/cygnus/release_name/build
% cd /usr/cygnus/release_name/build
% /usr/cygnus/release_name/src/configure
% --prefix=/opt/cygnus/release_name \
```

NOTE: `--target=` is only for cross development.`--target=` is unneccessary for *native configuration.* `configure` will default to native configuration when using `--target`.

Expect `configure` to take approximately 15-45 minutes to run, depending on the load of your build system, the toolchain being configured and the sources used.

2

Building and installing binaries

After you configure the toolchain, build and install the binary programs.

For single host-target builds, see “Single host-target builds” on page 8 for details.

For multiple host builds, see “Multiple host builds” on page 9 for details.

Some vendor-supplied `make` programs do not build the toolkit correctly, so for simplicity, use GNU `make` to rebuild from source. A precompiled copy of GNU `make` is in GNUPro Toolkit.

If you are not familiar with `make`, see “Overview of `make`” in *GNU Make* in ***GNUPro Utilities***.

Single host-target builds

In this example, a copy of GNU `make` is in `/usr/local/bin` and is called `make`.

1. Move to the build directory, using instructions like the following input.

   ```
   % cd /usr/cygnus/release_name/build
   ```

   Then, use the following instruction to run `make`.

   ```
   % /usr/local/bin/make all
   ```

2. This takes a while to complete. When `make all` finishes, install the tools with the following input.

   ```
   % /usr/local/bin/make install
   ```

3. When you have verified that your toolchain is functioning properly, you may remove the build directory to conserve disk space.

Multiple host builds

GNUPro Toolkit uses two options so that you can have a directory named `/usr/cygnus/`*`release_name`*, with multiple hosts and target versions in one place.

The host-specific files are in `/usr/cygnus/`*`release_name`*`/H-`*`hostspec`*`.` *`hostspec`* stands for the canonical name describing the host. For instance, for an Hitachi SH system, "`sh-hms-coff`" is the canonical name.

A compiler, for whatever target it's addressing, is a *host-specific* program, which means that it only runs on one host system. A help file is *host-independent* and, so, it is independent of its host system. So, imagine you have, for instance, Hitachi SH systems and IBM PowerPC running AIX 4.2 on one network. There is support for native compilers on both systems, and you want a single `/usr/cygnus` directory that can be NFS-mounted on all of your Windows NT machines. Use the following process.

1. Place all the programs that run on Hitachi SH systems in the following location.

   `/usr/cygnus/`*`release_name`*`/sh-hms-coff`

2. Place all the programs that run on PowerPC systems in the following location.

   `/usr/cygnus/`*`release_name`*`/powerpc-ibm-aix4.2`

   This shares the `man` pages, text configuration files, and other files for GNUPro Toolkit.

GNUPro Toolkit is designed for this kind of ***multiple-host*** environment. Set up the tools this way by adding a `–exec-prefix=` command line option to `configure` when configuring the toolkit.

3

Troubleshooting

The following documentation discusses warnings or error messages that may display during your build process. Each message has a troubleshooting approach accompanying it for resolution of the problem that you're addressing.

Error messages and warnings

The following warnings or error messages may display.

`Make: Fatal error: Don't know how to make target foo.c`
The most likely problem is that you are not using GNU **`make`**. Use the `--version` option for telling which version you are running; if you have this error message, run the command, `make –-version`. If you are not using GNU **`make`**, the `make` program will not recognize the `–-version` option.

`Incorrect compiler used.`
When `configure` runs, it looks for an appropriate compiler, first `gcc`, then `cc`. If neither of these is correct, specify the name of the compiler at configuration time. We recommend that you always build with `gcc`. Specify the compiler by setting the `$CC` environment variable before running `configure`. In the following example, the correct compiler is in `/usr/progressive/bin/gcc`.

- If you are using C shell, use a command similar to the following example's input (where */opt/vendor/bin/* is the path of the compiler, *not-your-usual-cc*).

```
% set CC/opt/vendor/bin/not-your-usual-cc
% configure ...
% make
```

- In the Bourne shell (`/bin/sh`) or the Korn shell, use the following example's input (where */opt/vendor/bin/* is the path of the compiler, *not-your-usual-cc*).

```
% CC=/opt/vendor/bin/not-your-usual-cc
% export CC
% configure ...
% make
```

If you still experience configuration problems, first try to rerun `configure` by adding the command line option, `--verbose`. It's best to redirect the output to a log file while running this process.

- If you are using C shell, use a command similar to the following example's input.

```
% configure --–verbose ... >& configure.out
```

- With Bourne shell, use the following example's input.

```
% configure --–verbose ... >configure.out 2>&1
```

Some seemingly unrelated problems arise after applying patches or making other changes. For instance, sometimes file dependencies get confused. With any trouble you have building, an easy step to take is to remove your build directory completely and then rebuild in an empty build direrctory, using input like the

following example (where *release_name* is the version name of the GNUPro Toolkit which you are using).

```
% rm -rf /usr/cygnus/release_name/build
% mkdir /usr/cygnus/release_name/build
```

For more information on installing GNUPro Toolkit, see *Installation* in ***Getting Started with GNUPro Toolkit***.

If it's not obvious which part of the toolkit is failing, check the last line in the log that begins "`Configuring...`" such as, with the debugging tool, **gdb**, you'd see "`Configuring gdb...`" before `configure` stops.

`configure` also creates a `config.log` file in each sub-directory in which it runs tests. Check the end of the `config.log` that failed for specific information about what went wrong. The last page (25-30 lines) of this file should be plenty, but if in doubt, send the entire file. For help, contact Cygnus using `send-pr`. For information on contacting Cygnus technical support, see "How to Contact Cygnus" on page iv.

Sending Cygnus your problem reports

If the cause of the problem is still not clear, send in a problem report. To send a report, use email: **`bugs@cygnus.com`**.

For a complete description of the automatic problem reporting system, see *Reporting Problems* in ***GNUPro Advanced Topics***.

See also “configure problem reporting” on page 15 and “build problem reporting” on page 16.

configure problem reporting

If your configuration problems are still confusing, send in a `configure` problem report. Send us a copy of the top-level `config.status` file. This file shows which command line options were used to configure the toolkit.

`config.status` (assuming the example pathnames we've used) should be in `/usr/cygnus/`*`release_name`*`/build/config.status`.

The `--verbose` option for `configure` produces the entire output from the last directory. For instance, if `configure` fails in the `gas` directory, send in everything after the line which reads "`Configuring gas...`" and, if in doubt, send us a copy of all the output generated by `configure --verbose`; so that Cygnus technical support staff can more easily determine the problem and quickly resolve it.

build problem reporting

If your build problems are still confusing, send in a build problem report.

IMPORTANT: Use "build" for your Problem Report category.

First, verify that you're using GNU make, using the process outlined in the examples with the "**Make: Fatal error: Don't know how to make target foo.c**" error (see "Error messages and warnings" on page 12).

We'll need the following information.

- Send us the top-level config.status file. This file will indicate which command line options were used to configure the toolkit. This file should be (using the example pathnames that we've used so far) at /usr/cygnus/*release_name*/build/config.status.
- Send us the output of the make command that is failing. Only the output from the last directory is probably useful, just as with a configure Problem Report. If in doubt, send the entire output from make and Cygnus technical support staff will determine the problem and resolve it

4

Patching

After a problem report (PR) has been submitted to Cygnus, and the corrective source code has been written and tested, compare the old and new contents of the source directory, using the Unix ‘`diff`’ command and the output is sent out as a “patch.”

To apply the patch to your source code, you will need to move to the source directory (`cd` into the `src`). The full default pathname is `/usr/cygnus/`*`release_name`*`/src`. *`release_name`* could be replaced with ‘`gnupro-98r1`’ if that is appropriate. Save the patch as a file, such as `/tmp/patch`, and run the `patch` program.

```
patch -p < /tmp/patch
```

You do not need to edit out all the non-patch text from the file, `/tmp/patch`. The `patch` program will recognize where the real patch begins.

IMPORTANT! Do not cut-and-paste the patch with a windowing system like X-Windows; tab characters are important and they are usually not preserved correctly when using cut-and-pasting methods.

See also “Invoking patch” on page 183 and “Options to patch” on page 189 in *Comparing and Merging Files*. If the patch is rejected, there will be a filename ending in ‘`.rej`’ in the source directory. For instance, if the patch was against the file ‘`src/gcc/reload.c`’, and the patch was rejected, the rejection would be called ‘`src/gcc/reload.c.rej`’. Although it will take a while to run, you can search all files

for a rejected patch with a command like the following example.

```
% find . -name '*.rej' -print
```

GNPRO TOOLKIT™

GNU Online Documentation

98r1
July, 1998

CYGNUS

Copyright © 1988-1998 Free Software Foundation

Permission is granted to make and distribute verbatim copies of this documentation provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the "GNU General Public License" are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

This document was written by Brian J. Fox: `bfox@ai.mit.edu`

This documentation has been prepared by Cygnus.

All rights reserved.

GNUPro™, the GNUPro™ logo, and the Cygnus logo are trademarks of Cygnus. All other brand and product names are trademarks of their respective owners.

To contact the Cygnus Technical Publications staff, email: `doc@cygnus.com`.

1

GNU online documentation overview

The GNU `info` program is online documentation used to view help files on an ASCII terminal. `info` files are the result of processing Texinfo files with the program, `Makeinfo`, using the Emacs editing command: M-x `texinfo-format-buffer`.

Texinfo is a documentation language allowing printed and online documentation (an **`info`** file) to be produced from a single source file. The following documentation discusses `info` in more detail.

- "Using info" on page 22
- "Reading info files" on page 23
- "Making info files from Texinfo files" on page 45

Using info

info is organized into *nodes*, corresponding to the chapters and sections of printed books. You can follow them in sequence just like in the printed book or, using menus, go quickly to the node having the information you need.

info has *hot* references; if one section refers to another, you can tell **info** to take you immediately to that other section. You can get back again easily to take up your reading where you left off. Naturally, you can also search for particular words and phrases.

The best way to get started with the online documentation system is to use a programmed tutorial by running `info` itself. You run `info` by just typing its name—no options or arguments are necessary—at your shell prompt (shown here as '`#`'), as in the following example.

```
# info
```

info displays its first screen, a menu of available documentation, and waits. To request a tutorial for learning **info**, type `h`. Or, from Emacs document browsing mode, in the window's command buffer, type `C-h, i`.

You can exit Info any time by typing `q`.

info also displays a summary of all its commands when you type `?`.

2

Reading `info` files

You can read the documentation for GNU software either on paper, as with any other instruction manual, or as online `info` files, using an ordinary ASCII terminal.

You can browse through the online documentation with GNU Emacs. The program, `info`, is a small program intended just for the purpose of viewing help files.

`info` files are generated by the program, **`Makeinfo`**, from a Texinfo source file.

Texinfo is a documentation markup language designed to allow the same source file to generate either printed or online documentation. The Texinfo language is described in ***Texinfo: The GNU Documentation Format***.

Command line options

GNU `info` accepts several options to control the initial node being viewed, and to specify which directories to search for `info` files. The following is a template showing an invocation of GNU `info` from the shell.

```
info [-- option-name option-value] menu-item ...
```

The following *`option-names`* are available when invoking `info` from the shell.

`--directory` *`directory-path`*
`-d` *`directory-path`*
: Adds *`directory-path`* to the list of directory paths searched when Info needs to find a file. You may issue `--directory` multiple times; once for each directory which contains info files.

  Alternatively, you may specify a value for the environment variable, `INFOPATH`; if `--directory` is not given, the value of `INFOPATH` is used. The value of `INFOPATH` is a colon separated list of directory names. If you do not supply `INFOPATH` or `--directory-path`, a default path is used.

`--file` *`filename`*
`-f` *`filename`*
: Specifies a particular `info` file to visit. Instead of visiting the file, `dir`, `info` will start with `(`*`filename`*`) Top` as the first file and node.

`--node` *`nodename`*
`-n` *`nodename`*
: Specifies a particular node to visit in the initial file loaded. This is especially useful in conjunction with `--file`[†].

  You can specify `--node` multiple times. For an interactive Info session, each *`nodename`* is visited in its own window. For a non-interactive Info (such as when `--output` is given), each *`node-name`* is processed sequentially.

`--output` *`filename`*
`-o` *`filename`*
: Specify *`filename`* as the name of a file to output to. Each node that Info visits will be output to *`filename`* instead of interactively viewed. A value of '-' for *`filename`* specifies the standard output.

`--subnodes`
: This option only has meaning given in conjunction with `--output`. It means to recursively output the nodes appearing in the menus of each node being output. Menu items which resolve to external info files are not output, and neither are menu items which are members of an index. Each node is only output once.

[†] Of course, you can specify both the file and node in a `--node` command; but don't forget to escape the open and close parentheses from the shell as in: `info --node '(emacs)Buffers'`

`--help`

`-h`

Produces a relatively brief description of the available Info options.

`--version`

Prints the version information of `info` and exits.

`menu-item`

Remaining arguments to `info` are treated as the names of menu items. The first argument would be a menu item in the initial node visited, while the second argument would be a menu item in the first argument's node. You can easily move to the node of your choice by specifying the menu names which describe the path to that node, as in the following example.

```
info emacs buffers
```

This input example selects the menu item '`Emacs`' in the node '`(dir)Top`', and then selects the menu item '`Buffers`' in the node '`(emacs)Top`'.

Moving the cursor

Many people find that reading screens of text page by page is made easier when one is able to indicate particular pieces of text with some kind of pointing device. Since this is the case, GNU `info` (both the Emacs and standalone versions) have several commands which let you move the cursor about the screen. The notation in this documentation to describe keystrokes is identical to the notation used within the Emacs manual, and the GNU Readline manual. See "Characters, Keys and Commands" in ***GNU Emacs Manual***, if you are unfamiliar with the notation.

The following table lists the basic `info` cursor movement commands.

Each entry consists of the key sequence to type to perform the cursor movement: a command like M-x[†], and a short description of what it does.

Cursor motion also uses a *numeric* argument; for further discussion, see "Miscellaneous commands" on page 39. A numeric argument executes a command that many times; for example, `4` given to `next-line` moves the cursor *down* 4 lines.

A negative numeric argument reverses the motion; thus, an argument of `-4` for the `next-line` command moves the cursor *up* 4 lines.

C-n (`next-line`)
: Moves the cursor down to the next line.

C-p (`prev-line`)
: Move the cursor up to the previous line.

C-a (`beginning-of-line`)
: Move the cursor to the start of the current line.

C-e (`end-of-line`)
: Moves the cursor to the end of the current line.

C-f (`forward-char`)
: Move the cursor forward a character.

C-b (`backward-char`)
: Move the cursor backward a character.

M-f (`forward-word`)
: Moves the cursor forward a word.

M-b (`backward-word`)
: Moves the cursor backward a word.

M-< (`beginning-of-node`)

[†] M-x is also a command; it invokes `execute-extended-command`. See "Keyboard Input" in the ***GNU Emacs Manual*** for more detailed information.

b
: Moves the cursor to the start of the current node.

M-> (`end-of-node`)
: Moves the cursor to the end of the current node.

M-r (`move-to-window-line`)
: Moves the cursor to a specific line of the window. Without a numeric argument, M-r moves the cursor to the start of the line in the center of the window. With a numeric argument of *n*, M-r moves the cursor to the start of the *n*th line in the window.

Moving text within a window

Sometimes you are looking at a full screen of text, and only part of the current paragraph you are reading is visible on the screen. The commands detailed in the following section are used to shift which part of the current node is visible on the screen.

SPC / Spacebar (`scroll-forward`)

C-v
: Shifts the text in this window up to show more of the node which is currently below the bottom of the window. A numeric argument shows that many more lines at the bottom of the window; a numeric argument of 4 shifts all text in the window up 4 lines (discarding the top 4 lines), and shows four new lines at the bottom of the window. Without a numeric argument, Spacebar takes the bottom two lines of the window and places them at the top of the window, redisplaying almost a completely new screenful of lines.

Del /Delete (`scroll-backward`)

M-v
: Shifts the text in this window down, the inverse of scroll-forward.

  `scroll-forward` and `scroll-backward` moves forward and backward through the node structure of the file. If you press Spacebar while viewing the end of a node, or Del while viewing the beginning of a node, what happens is controlled by the variable `scroll-behavior`. See “Manipulating Variables,” page Manipulating variables for more information.

C-l (`redraw-display`)
: Redraws the display from scratch, or shifts the line whit the cursor to a specified location. With no numeric argument, ‘C-l’ clears the screen, and then redraws its entire contents. With a numeric argument of *n*, the line with the cursor is shifts to the *nth* line of the window.

C-x w (`toggle-wrap`)
: Toggles the state of line wrapping in the current window. Normally, when lines *wrap* when they are longer than the screen width,; i.e., they continue on the next line. Lines which wrap display a ‘\’ in the rightmost column of the screen. You can cause such lines to be terminated at the rightmost column by changing the state of line wrapping in the window with C-x, w. When a line contains more than one screen width, ‘$’ appears in the rightmost column of the line, and the remainder is invisible.

Selecting a new node

The following documentation details the numerous `info` commands which select a new node to view in the current window.

The most basic node commands are ‘n’, ‘p’, ‘u’, and ‘l’.

When you are viewing a node, the top line of the node contains some `info` *pointers* which describe where the *next*, *previous*, and *up* nodes are. `info` uses this line to move about the node structure of the file when you type the following commands.

`n` (`next-node`)
: Selects the `Next` node.

`p` (`prev-node`)
: Selects the `Prev` (previous) node.

`u` (`up-node`)
: Selects the `Up` node.

You can easily select a node that you have already viewed in this window by using the ‘l’ command (this name stands for *last*), to actually move through the list of already visited nodes for this window. ‘l’ with a negative numeric argument moves forward through the history of nodes for this window, so you can quickly step between two adjacent (in viewing history) nodes.

`l` (`history-node`)
: Selects the most recently selected node in this window.

Two additional commands make it easy to select the most commonly selected nodes; they are ‘t’ and ‘d’.

`t` (`top-node`)
: Selects the node `Top` in the current info file.

`d` (`dir-node`)
: Selects the directory node (i.e., the node, `(dir)`).

The following are some other commands which immediately result in the selection of a different node in the current window.

`<` (`first-node`)
: Selects the first node which appears in this file. This node is most often ‘Top’, but it doesn’t have to be.

`>` (`last-node`)
: Selects the last node which appears in this file.

`]` (`global-next-node`)
: Moves forward or down through node structure. If the node that you are currently viewing has a ‘`Next`’ pointer, that node is selected. Otherwise, if this node has a

menu, the first menu item is selected. If there is no ‘Next’ and no menu, the same process is tried with the ‘Up’ node of this node.

[(global-prev-node)
Moves backward or up through node structure. If the node that you are currently viewing has a ‘Prev’ pointer, that node is selected. Otherwise, if the node has an ‘Up’ pointer, that node is selected, and if it has a menu, the last item in the menu is selected.

global-next-node and global-prev- node behave the same as simply scrolling through the file with Spacebar and Del; see scroll-behavior in “Manipulating variables” on page 41 for more information.

g (goto-node)
Reads the name of a node and selects it. No completion is done while reading the node name, since the desired node may reside in a separate file. The node must be typed exactly as it appears in the info file. A file name may be included as with any node specification, as in the following example.

```
g(emacs)Buffers
```

This input finds the ‘Buffers’ node in the ‘emacs’ info file.

C-x, k (kill-node)
Kills a node. The node name is prompted for in the echo area, with a default of the current node. *Killing* a node means that info tries hard to forget about it, removing it from the list of history nodes kept for the window where that node is found. Another node is selected in the window which contained the killed node.

C-x, C-f (view-file)
Reads the name of a file and selects the entire file.

C-x, C-f *filename*, is equivalent to typing g(*filename*)*

C-x, C-b (list-visited-nodes)
Makes a window containing a menu of all of the currently visited nodes. This window becomes the selected window, and you may use the standard info commands within it.

C-x, b (select-visited-node)
Selects a node which has been previously visited in a visible window. This is similar to C-x, C-b followed by ‘m’, but no window is created.

Searching an `info` file

GNU **`info`** allows you to search for a sequence of characters throughout an entire info file, search through the indices of an `info` file, or find areas within an `info` file which discuss a particular topic.

s (`search`)
: Reads a string in the echo area and searches for it.

C-s (`isearch-forward`)
: Interactively searches forward through the info file for a string you type.

C-r (`isearch-backward`)
: Interactively searches backward through the info file for a string as you type it.

i (`index-search`)
: Looks up a string in the indices for this info file, and selects the node that the found index entry points to.

, (`next-index-match`)
: Moves to the node containing the next matching index item from the last ‘i’ command.

The most basic searching command is ‘`s`’ (`search`). The ‘`s`’ command prompts you for a string in the echo area, and then searches the remainder of the **`info`** file for an occurrence of that string. If the string is found, the node containing it is selected, and the cursor is left positioned at the start of the found string. Subsequent ‘`s`’ commands show you the default search string within ‘[’ and ‘]’; pressing Enter, instead of typing a new string will use the default search string.

Incremental searching is similar to basic searching, but the string is looked up while you are typing it, instead of waiting until the entire search string has been specified.

Selecting cross references

We have already discussed the ‘`Next`’, ‘`Prev`’, and ‘`Up`’ pointers which appear at the top of a node. In addition to these pointers, a node may contain other pointers which refer you to a different node, perhaps in another **`info`** file. Such pointers are called *cross references*, or *xrefs* for short.

Parts of an xref

Cross references have two major parts: the first part is called the *label*; it is the name that you can use to refer to the cross reference, and the second is the *target*; it is the full name of the node to which the cross reference points.

The target is separated from the label by a colon ‘`:`’; first, the label appears, and then the target. For instance, the following example’s input shows a cross reference menu, where the single colon separates the label from the target.

```
* Foo Label: Foo Target. More information about Foo.
```

The ‘`.`’ is not part of the target; it serves only to let **`info`** know where the target name ends.

A shorthand way of specifying references allows two adjacent colons to stand for a target name, as in the following example.

```
* Foo Commands:: Commands pertaining to Foo.
```

In the previous example, the name of the target is the same as the name of the label, in this case `Foo Commands`.

You will normally see two types of cross references while viewing nodes: *menu* references, and *note* references. Menu references appear within a node’s menu; they begin with a ‘`*`’ at the beginning of a line, and continue with a label, a target, and a comment which describes what the contents of the node pointed to contains.

NOTE: References appear within the body of the node text; they begin with `*Note`, and continue with a label and a target.

Like ‘`Next`’, ‘`Prev`’ and ‘`Up`’ pointers, cross references can point to any valid node. They are used to refer you to a place where more detailed information can be found on a particular subject.

See “Cross References” in ***Texinfo, The GNU Documentation Format***, for more information on creating your own Texinfo cross references.

Selecting xrefs

The following lists the `info` commands that operate on menu items.

1 (`menu-digit`)
2 ...9
: Within an **`info`** window, pressing a single digit, (such as '1'), selects that menu item, and places its node in the current window. For convenience, there is one exception; pressing '0' selects the *last* item in the node's menu.

0 (`last-menu-item`)
: Select the last item in the current node's menu.

m (`menu-item`)
: Reads the name of a menu item in the echo area and selects its node. Completion is available while reading the menu label.

M-x `find-menu`
: Moves the cursor to the start of this node's menu.

The following lists the `info` commands which operate on note cross references.

f (`xref-item`)
r
: Reads the name of a note cross reference in the echo area and selects its node. Completion is available while reading the cross reference label.

Finally, the next few commands operate on both menu or note references.

Tab (`move-to-next-xref`)
: Moves the cursor to the start of the next nearest menu item or note reference in the current node. You can also then use the following command, Return (`select-reference- this-line`), to select the menu or note reference.

M-Tab (`move-to-prev-xref`)
: Moves the cursor to the start of the nearest previous menu item or note reference in the current node.

Enter / Return(`select-reference-this-line`)
: Selects the menu item or note reference appearing on the line where the cursor currently is.

Manipulating multiple windows

A *window* is a place to show the text of a node. Windows have a view area where the text of the node is displayed, and an associated *mode* line, which briefly describes the node being viewed.

GNU `info` supports multiple windows appearing in a single screen; each window is separated from the next by its modeline. At any time, there is only one *active* window, that is, the window in which the cursor appears. There are commands available for creating windows, changing the size of windows, selecting which window is active, and for deleting windows.

The mode line

A *mode* line is a line of inverse video which appears at the bottom of an info window. It describes the contents of the previously displayed window; this information includes the name of the file and node appearing in that window, the number of screen lines it takes to display the node, and the percentage of text that is above the top of the window. It can also tell you if the indirect tags table for this `info` file needs to be updated, and whether or not the info file was compressed when stored on disk. The following is a sample mode line for a window containing an uncompressed file named ‘`dir`’, showing the node ‘`Top`’.

```
-----Info: (dir)Top, 40 lines --Top-------------------------
            ^^^   ^^^             ^^
```

```
           (file)Node #lines where
```

When a node comes from a file which is compressed on disk, this is indicated in the mode line with two small ‘`z`’s. In addition, if the `info` file containing the node has been split into subfiles, the name of the subfile containing the node appears in the modeline as well.

```
--zz-Info: (emacs)Top, 291 lines --Top-- Subfile: emacs-1.Z-
```

When `info` makes a node internally, such that there is no corresponding `info` file on disk, the name of the node is surrounded by asterisks (‘`*`’). The name itself tells you what the contents of the window are; the following sample mode line shows an internally constructed node showing possible one possible completion.

```
-----Info: *Completions*, 7 lines --All--------------------
```

Window commands

To view more than one node at a time, **`info`** can display more than one window. Each window has its own mode line (see “The mode line” on page 34) and history of nodes viewed in that window (for information on `history-node`, see “Selecting a new node” on page 29).

C-x, o (`next-window`)
: Selects the next window on the screen. The echo area can *only* be selected if it is already in use, and you have left it temporarily. Normally, C-x, o simply moves the cursor into the next window on the screen, or if you are already within the last window, into the first window on the screen. Given a numeric argument, C-x, o moves over that many windows. A negative argument causes C-x, o to select the previous window on the screen.

M-x (`prev-window`)
: Selects the previous window on the screen. This is identical to C-x, o with a negative argument.

C-x, 2 (`split-window`)
: Splits the current window into two windows, both showing the same node. Each window is one half the size of the original window, and the cursor remains in the original window. The variable, `automatic-tiling`, can cause all of the windows on the screen to be resized for you automatically; for more information on `automatic-tiling`, see “Manipulating variables” on page 41.

C-x, 0 (`delete-window`)
: Deletes the current window from the screen. If you have made too many windows and your screen appears cluttered, this is the way to get rid of some of them.

C-x, 1 (`keep-one-window`)
: Deletes all of the windows excepting the current one.

Esc C-v (`scroll-other-window`)
: Scrolls the other window, in the same fashion that ‘C-v’ might scroll the current window. Given a negative argument, the *other* window is scrolled backward.

C-x,^ (`grow-window`)Grows (or shrinks) the current window. Given a numeric argument, grows the current window that many lines; with a negative numeric argument, the window is shrunk instead.

C-x, t (`tile-windows`)
: Divides the available screen space among all of the visible windows. Each window is given an equal portion of the screen in which to display its contents. The variable `automatic-tiling` can cause `tile-windows` to be called when a window is created or deleted. For more information on `automatic-tiling`, see “Manipulating variables” on page 41.

The Echo Area

The *echo area* is a one line window which appears at the bottom of the screen. It is used to display informative or error messages, and to read lines of input from you when that is necessary. Almost all of the commands available in the echo area are identical to their Emacs counterparts, so please refer to GNU Emacs documentation

for greater depth of discussion on the concepts of editing a line of text.

The following briefly details the commands that are available while input is being read in the echo area.

C-f (`echo-area-forward`)
: Moves forward a character.

C-b (`echo-area-backward`)
: Moves backward a character.

C-a (`echo-area-beg-of-line`)
: Moves to the start of the input line.

C-e (`echo-area-end-of-line`)
: Moves to the end of the input line.

M-f (`echo-area-forward-word`)
: Moves forward a word.

M-b (`echo-area-backward-word`)
: Moves backward a word.

Cd (`echo-area-delete`)
: Deletes the character under the cursor.

Del (`echo-area-rubout`)
: Deletes the character behind the cursor.

C-g (`echo-area-abort`)
: Cancels or quits the current operation. If completion is being read, C-g discards the text of the input line which does not match any completion. If the input line is empty, C-g aborts the calling function.

RET (`echo-area-newline`)
: Accepts (or forces completion of) the current input line.

C-q (`echo-area-quoted-insert`)
: Inserts the next character verbatim; for example, so you can insert control characters into a search string.

`printing character` (`echo-area-insert`)
: Inserts the character.

M-Tab (`echo-area-tab-insert`)
: Inserts a Tab character.

Ctrl-t (`echo-area-transpose-chars`)
: Transposes the characters at the cursor.

The next group of commands deal with killing and yanking text. For an in depth discussion of killing and yanking, see “Killing and Moving Text” in the ***GNU Emacs Manual***.

M-d (`echo-area-kill-word`)
: Kills the word following the cursor.

M-Del (`echo-area-backward-kill-word`)
: Kills the word preceding the cursor.

C-k (`echo-area-kill-line`)
: Kills the text from the cursor to the end of the line.

C-x, Del (`echo-area-backward-kill-line`)
: Kills the text from the cursor to the beginning of the line.

C-y (`echo-area-yank`)
: Yanks back the contents of the last kill.

M-y (`echo-area-yank-pop`)
: Yanks back a previous kill, removing the last yanked text first.

Sometimes when reading input in the echo area, the command that needed input will only accept one of a list of several choices. The choices represent the *possible completions*, and you must respond with one of them. Since there are a limited number of responses you can make, `info` allows you to abbreviate what you type, only typing as much of the response as is necessary to uniquely identify it. In addition, you can request `info` to fill in as much of the response as is possible; this is called *completion*.

The following commands are available when completing in the echo area.

Tab (`echo-area-complete`)
SPACEBAR
: Inserts as much of a completion as is possible.

? (`echo-area-possible-completions`)
: Displays a window containing a list of the possible completions of what you have typed so far. For example, say the available choices are the following if you typed an ‘f’, followed by ‘?’.

```
bar foliate
food forget
```

Possible completions would contain the choices which begin with ‘**f**’.

```
foliate food forget
```

Pressing Spacebar or Tab would result in ‘**fo**’ appearing in the echo area, since all of the choices which begin with ‘**f**’ continue with ‘**o**’. Now, typing ‘l’ followed by pressing Tab results in ‘**foliate**’ appearing in the echo area, since that is the only choice which begins with ‘**fol**’.

Esc Ctrl-v (`echo-area-scroll-completions-window`)
: Scrolls the completions window, if that is visible, or, if not, the *other* window.

Printing out nodes

You may wish to print out the contents of a node as a quick reference document for later use. `info` provides you with a command for printing.

In general, we recommend that you use the C program utility, `Makeinfo`, to create an `info` file from a Texinfo source file and then, by using the command, `texify`, format the document and print the DVI (*Device Independent*) file.

See ***Texinfo, The GNU Documentation Format*** manual for more details.

`info` also provides you with a command for printing.

M-x `print-node`
: Pipes the contents of the current node through the command in the environment variable, `INFO_PRINT_COMMAND`. If the variable doesn't exist, the node is simply piped to `lpr`.

Miscellaneous commands

GNU `info` contains several commands which self-document GNU `info` as the following discussions help to clarify.

M-x `describe-command`
: Reads the name of an `info` command in the echo area and then displays a brief description of what that command does.

M-x `describe-key`
: Reads a key sequence in the echo area, and then displays the name and documentation of the **`info`** command which a given key sequence invokes.

M-x `describe-variable`
: Reads the name of a variable in the echo area and then displays a brief description of what the variable affects.

M-x `where-is`
: Reads the name of an **`info`** command in the echo area, and then displays a key sequence which can be typed in order to invoke that command.

C-h (`get-help-window`)

?
: Creates (or moves into) the window displaying **`*Help*`**, and places a node containing a quick reference card into it. This window displays the most concise information about GNU **`info`** available.

h (`get-info-help-node`)
: Tries hard to visit the node **`(info)Help`**. The **`info`** file, '**`info.texi`**,' distributed with GNU **`info`**, contains this node. Of course, the file must first be processed with **`makeinfo`**, and then placed into the location of your info directory.

The following are the commands for creating a numeric argument.

C-u (`universal-argument`)
: Starts (or multiplies by 4) the current numeric argument. 'C-u' is a good way to give a small numeric argument to cursor movement or scrolling commands.

  C-u, C-v scrolls the screen 4 lines, while 'C-u, C-u, C-n' moves the cursor down 16 lines.

M-1 (`add-digit-to-numeric-arg`)

M-2**...** M-9
: Adds the digit value of the invoking key to the current numeric argument. Once **`info`** is reading a numeric argument, you may just type the digits of the argument, without the M prefix. For example, you might give C-l a numeric argument of `32` by using the keystroke sequence, C-u, 3, 2, C-l or M-3, 2, C-l.

  C-g is used to abort the reading of a multi-character key sequence, to cancel

lengthy operations (such as multi-file searches) and to cancel reading input in the echo area.

C-g (`abort-key`)
: Cancels current operation.

q (`quit`)
: Exits **`info`**.

If the operating system tells **`info`** that the screen is 60 lines tall, and it is actually only 40 lines tall, the following is a way to tell **`info`** that the operating system is correct.

M-x `set-screen-height`
: Reads a height value in the echo area and sets the height of the displayed screen to that value.

Finally, **`info`** provides a convenient way to display footnotes which might be associated with the current node that you are viewing:

Esc C-f (`show-footnotes`)
: Shows the footnotes (if any) associated with the current node in another window. You can have **`info`** automatically display the footnotes associated with a node when the node is selected by setting the variable, `automatic-footnotes`; for more information on `automatic-footnotes`, see “Manipulating variables” on page 41.

Manipulating variables

GNU **info** contains several variables whose values are looked at by various **info** commands. You can change the values of these variables, and thus change the behavior of **info** to more closely match your environment and `info` file reading manner.

M-x `set-variable`
: Reads the name of a variable, and the value for it, in the echo area and then sets the variable to that value. Completion is available when reading the variable name; often, completion is available when reading the value to give to the variable, but that depends on the variable itself. If a variable does not supply multiple choices to complete over, it expects a numeric value.

M-x `describe-variable`
: Reads the name of a variable in the echo area and then displays a brief description of what the variable affects.

What follows is a list of the variables that you can set in **info**.

`automatic-footnotes`
: When set to `On`, footnotes appear and disappear automatically. This variable is `On` by default. When a node is selected, a window containing the footnotes which appear in that node is created, and the footnotes are displayed within the new window. The window that **info** creates to contain the footnotes is called '`*Footnotes*`'. If a node is selected which contains no footnotes, and a '`*Footnotes*`' window is on the screen, the '`*Footnotes*`' window is deleted. Footnote windows created in this fashion are not automatically tiled so that they can use as little of the display as is possible.

`automatic-tiling`
: When set to `On`, creating or deleting a window *resizes* other windows. This variable is `Off` by default. Normally, typing 'Ctrl-x, 2' divides the current window into two equal parts. When `automatic-tiling` is set to `On`, all of the windows are resized automatically, keeping an equal number of lines visible in each window. There are exceptions to the automatic tiling; specifically, the windows '`*Completions*`' and '`*Footnotes*`' are *not* resized through automatic tiling; they remain their original size.

`visible-bell`
: When set to `On`, GNU **info** attempts to flash the screen instead of ringing the bell. This variable is `Off` by default.

  Of course, **info** can only flash the screen if the terminal allows it; in the case that the terminal does not allow it, the setting of this variable has no effect.

  However, you can set the `errors-ring-bell` variable to `Off` to make Info

perform quietly.

`errors-ring-bell`
When set to `On`, errors cause the bell to ring. The default setting of this variable is `On`.

`gc-compressed-files`
When set to `On`, **`info`** garbage collects files which had to be uncompressed. The default value of this variable is `Off`. Whenever a node is visited in **`info`**, the **`info`** file containing that node is read into core, and **`info`** reads information about the tags and nodes contained in that file. Once the tags information is read by **`info`**, it is never forgotten. However, the actual text of the nodes does not need to remain in core unless a particular **`info`** window needs it. For non-compressed files, the text of the nodes does not remain in core when it is no longer in use. But decompressing a file can be a time consuming operation, and so **`info`** tries hard not to do it twice. `gc-compressed-files` tells **`info`** it is okay to garbage collect the text of the nodes of a file which was compressed on disk.

`show-index-match`
When set to `On`, the portion of the matched search string is highlighted in the message which explains where the matched search string was found. The default value of this variable is `On`. When **`info`** displays the location where an index match was found, (for more information on `next-index-match` , see “Searching an info file” on page 31), the portion of the string that you had typed is highlighted by displaying it in the inverse case from its surrounding characters.

`scroll-behaviour`
Controls what happens when forward scrolling is requested at the end of a node, or when backward scrolling is requested at the beginning of a node. The default value for this variable is `Continuous`. There are three possible values for this variable:

`Continuous`
Tries to get the first item in this node’s menu, or failing that, the ‘`Next`’ node, or failing that, the ‘`Next`’ of the ‘`Up`’. This behavior is identical to using the ‘]’ (`global-next-node`) and ‘[’ (`global-prev- node`) commands.

`Next Only`
Only tries to get the ‘`Next`’ node.

`Page Only`
Simply gives up, changing nothing. If `scroll-behaviour` is `Page Only`, no scrolling command can change the node that is being viewed.

`scroll-step`
The number of lines to scroll when the cursor moves out of the window. Scrolling happens automatically if the cursor has moved out of the visible portion of the node text when it is time to display. Usually the scrolling is done so as to put the

cursor on the center line of the current window. However, if the variable `scroll-step` has a nonzero value, **`info`** attempts to scroll the node text by that many lines; if that is enough to bring the cursor back into the window, that is what is done. The default value of this variable is `0`, thus placing the cursor (and the text it is attached to) in the center of the window. Setting this variable to `1` causes a kind of *smooth scrolling* which some people prefer.

`ISO-Latin`
When set to `On`, **`info`** accepts and displays ISO Latin characters. By default, Info assumes an ASCII character set. `ISO-Latin` tells **`info`** that it is running in an environment where the European standard character set is in use, and allows you to input such characters to **`info`**, as well as display them.

3

Making `info` files from Texinfo files

`Makeinfo` is the program that builds **`info`** files from Texinfo files. Before reading this documentation, you should be familiar with reading **`info`** files. If you want to run `Makeinfo` on a Texinfo file prepared by someone else, this documentation contains most of what you need to know. However, to write your own Texinfo files, you should also read ***Texinfo, The GNU Documentation***.

Controlling paragraph formats

In general, `Makeinfo` *fills* the paragraphs that it outputs to the **`info`** file. Filling is the process of breaking up and connecting lines such that the output is nearly justified. With `Makeinfo`, you can control the following.

- The width of each paragraph (the *fill-column*).
- The amount of indentation that the first line of the paragraph receives (the *paragraph-indentation*).

Command line options for `makeinfo`

The following command line options are available for `Makeinfo`.

`-I` *dir*
: Adds *dir* to the directory search list for finding files which are included with the `@include` command. By default, only the current directory is searched.

`-D` *var*
: Defines the `texinfo` flag, *var*. This is equivalent to '`@set` *var*' in the Texinfo file.

`-U` *var*
: Makes the Texinfo flag, *var*, undefined. This is equivalent to '`@clear` *var*' in the Texinfo file.

`--error-limit num`
: Sets the maximum number of errors that `Makeinfo` will print before exiting (on the assumption that continuing would be useless). The default number of errors printed before `Makeinfo` gives up on processing the input file is 100.

`--fill-column num`
: Specifies the maximum right-hand edge of a line. Paragraphs that are filled will be filled to this width. The default value for fill-column is 72.

`--footnote-style style`
: Sets the footnote style to `style`. `style` should either be '`separate`' to have `Makeinfo` create a separate node containing the footnotes which appear in the current node, or '`end`' to have `Makeinfo` place the footnotes at the end of the current node.

`--no-headers`
: Suppress the generation of menus and node headers. This option is useful together with the '`--output file`' and '`--no-split`' options (see following options) to produce a simple formatted file (suitable for printing on a dumb printer) from Texinfo source. If you do not have TEX, these two options may allow you to get readable hard copy.

`--no-split`
: Suppress the splitting stage of `Makeinfo`. In general, large output files (where the size is greater than 70k bytes) are split into smaller subfiles, each one approximately 50k bytes.

  If you specify '`--no-split`', `Makeinfo` will not split up the output file.

`--no-pointer-validate`
`--no-validate`
: Suppress the validation phase of `Makeinfo`. Normally, after the file is processed, some consistency checks are made to ensure that cross references can be resolved, and so forth. See "What makes a valid info file?" on page 49.

`--no-warn`
: Suppress the output of warning messages. This does not suppress the output of error messages, simply warnings. You might want this if the file you are creating has texinfo examples in it, and the nodes that are referenced don't actually exist.

`--no-number-footnotes`
: Suppress the automatic numbering of footnotes. The default is to number each footnote sequentially in a single node, resetting the current footnote number to `1` at the start of each node.

`--output` *`file`*
`-o` *`file`*
: Specify that the output should be directed to *`file`* instead of the file name specified in the `@setfilename` command found in the Texinfo source. *`file`* can be the special token '`-`', which specifies standard output.

`--paragraph-indent` *`num`*
: Sets the paragraph indentation to *`num`*. The value of *`num`* is interpreted as follows:

  - A value of `0` (or 'none') means not to change the existing indentation (in the source file) at the start of paragraphs.
  - A value less than zero means to indent paragraph starts to column zero by deleting any existing indentation.
  - A value greater than zero is the number of spaces to leave at the front of each paragraph start.

`--reference-limit` *`num`*
: When a node has many references in a single Texinfo file, this may indicate an error in the structure of the file. *`num`* is the number of times a given node may be referenced before `Makeinfo` prints a warning message about it (with `@prev`, `@next`, or `@note` appearing in an `@menu`, for example).

`--verbose`
: Causes `Makeinfo` to inform you as to what it is doing. Normally `Makeinfo` only outputs text if there are errors or warnings.

`--version`
: Displays the `Makeinfo` version number.

What makes a valid `info` file?

If you have not used '`--no-pointer-validate`' to suppress validation, `Makeinfo` will check the validity of the final info file. Mostly, this means ensuring that nodes you have referenced really exist. What follows is a complete list of what is checked.

- If a node reference such as `Prev`, `Next` or `Up` is a reference to a node in this file (meaning that it is not an external reference such as '`(DIR)`'), then the referenced node must exist.
- In a given node, if the node referenced by the `Prev` is different than the node referenced by the `Up`, then the node referenced by the `Prev` must have a `Next` which references this node.
- Every node except `Top` must have an `Up` field.
- The node referenced by `Up` must contain a reference to this node, other than a `Next` reference. Obviously, this includes menu items and followed references.
- If the `Next` reference is not the same as the `Next` reference of the `Up` reference, then the node referenced by `Next` must have a `Prev` reference pointing back at this node. This rule still allows the last node in a section to point to the first node of the next chapter.

Defaulting the Prev, Next, and Up

If you write the @node commands in your Texinfo source file without Next, Prev, and Up pointers, Makeinfo will fill in the pointers from context (by reference to the menus in your source file). Although the definition of an info file allows a great deal of flexibility, there are some conventions that you are urged to follow. By letting Makeinfo default the Next, Prev, and Up pointers you can follow these conventions with a minimum of effort.

A common error occurs when adding a new node to a menu; often the nodes which are referenced in the menu do not point to each other in the same order as they appear in the menu.

Makeinfo node defaulting helps with this particular problem by not requiring any explicit information beyond adding the new node (so long as you do include it in a menu). The node to receive the defaulted pointers must be followed immediately by a sectioning command, such as @chapter or @section, and must appear in a menu that is one sectioning level or more above the sectioning level that this node is to have.

What follows is an example of how to use this feature.

```
@setfilename default-nodes.info
@node Top
@chapter Introduction
@menu
* foo:: the foo node
* bar:: the bar node
@end menu
@node foo
@section foo
this is the foo node.
@node bar
@section Bar
This is the Bar node.
@bye
```

The previous input produces the following output.

```
Info file default-nodes.info, produced by Makeinfo, -*- Text -*-

from input file default-nodes.texinfo.

File: default-nodes.info, Node: Top

Introduction ************
* Menu:

* foo:: the foo node
* bar:: the bar node
```

```
File: default-nodes.info, Node: foo, Next: bar, Up: Top
foo
===

this is the foo node.

File: default-nodes.info, Node: bar, Prev: foo, Up: Top

Bar
===

This is the Bar node.
```

GNPRO TOOLKIT™

Reporting Problems

98r1
July, 1998

CYGNUS

Copyright © 1988-1998 Cygnus

All rights reserved.

GNUPro™, the GNUPro™ logo, and the Cygnus logo are trademarks of Cygnus. All other brand and product names are trademarks of their respective owners.

Permission is granted to make and distribute verbatim copies of this documentation provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the “GNU General Public License” are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

This documentation has been prepared by Cygnus.

To contact the Cygnus Technical Publications staff, email: `doc@cygnus.com`.

1

Introduction to `send-pr`

With GNUPro Toolkit, users have `send-pr` available to submit support questions and problem reports to a central Support Site with their electronic mail. The following documentation discusses managing the submission of reports.

`send-pr` is part of a collection of programs known collectively as PRMS, the GNU Problem Report Management System. PRMS consists of several programs that formulate and partially administer a database of *Problem Reports*, or *PR*s, at a central Support Site. A PR goes through several states in its lifetime; PRMS tracks the PR and all information associated with it through each state and acts as an archive for PRs which have been closed.

In general, you can use any editor and mailer to submit valid Problem Reports, as long as the format required by PRMS is preserved. `send-pr` automates the process, however, ensuring that certain fields necessary for automatic processing are present.

`send-pr` is strongly recommended for all initial problem-oriented correspondence with your Support Site. The organization you submit Problem Reports to supplies an address to which further information can be sent; the person responsible for the

category of the problem you report directly contacts you.

Because `send-pr` exists as a shell script (in `/bin/sh`) and as an Elisp file for use with a text editor like GNU Emacs, it can be used from any machine on a network that is running a shell script and/or Emacs.

Cygnus uses PRMS and `send-pr` extensively for their support activities, finding problem tracking to be a crucial part of everyday business.

Using electronic mail, customers can communicate their problems effectively to Cygnus and then automatically receive confirmation and notification of changes regarding the status of the problems that they reported.

Cygnus supports the GNU compiling tools on over 150 native and cross-platform environments. GNUPro Toolkit contains PRMS and `send-pr`.

2

Installing `send-pr` on your system

If you receive `send-pr` as part of the GNUPro Toolkit software, it installs with the rest of the distribution. If you are using PRMS at your site as well, you must decide where `send-pr` sends Problem Reports by default; see “Setting a default site” on page 58 and “Installing send-pr by itself” on page 59.

Setting a default *site*

`send-pr` is capable of sending Problem Reports to any number of Support Sites, using mail aliases which have '`-prms`' appended to them. `send-pr` automatically appends the suffix, so that when you type `send-pr` *`site`*, the Problem Report goes to the address noted in the '`aliases`' file as '*`site`*`-prms`' (where *`site`* is the designation for your site). You can do this in the Emacs version of `send-pr` by invoking the program with the keystroke combination, C-u, M-x, and typing `send-pr` in the buffer.

You are prompted for *`site`*. *`site`* is also used to error-check the '**`>Category:`**' field, as a precaution against sending mistaken information (and against sending information to the wrong site). You may also simply type `send-pr` from the shell (or using the keystroke combination, M-x, and typing `send-pr` in the buffer), and the Problem Report that you generate will be sent to the site, usually the site from which you received your distribution of `send-pr`. If you use PRMS at your own organization, the default is usually your local address for reporting problems. To change this, simply edit the file, `Makefile`, *before* installing and change the line, `PRMS_SITE =`*`site`*, to reflect the *`site`* where you wish to send PRs by default.

Installing send-pr by itself

Install send-pr by using the following steps (you may need root access in order to change the 'aliases' file and to install send-pr):

1. Unpack the distribution into a directory which we refer to as srcdir.
2. Edit the file 'Makefile' to reflect local conventions. Specifically, you should edit the variable, '*prefix*', to alter the installation location.

   The default is '/usr/local'. All files are installed under '*prefix*' (see the following discussion).
3. Run, using the following command.

   ```
   make all install [ info ] [ install-info ] [ clean ]
   ```

   The targets mean the following:

   - all
     Builds send-pr and install-sid
   - install
     Installs the following:
     - install-sid
     - send-pr into '*prefix*/bin'

       send-pr.[†] into '*prefix*/man/man1'
     - *site* is the list of valid categories for the Support Site from which you received send-pr. This installs as '*prefix*/lib/prms/*site*'
     - send-pr.el into ' *prefix*/lib/emacs/lisp'
   - info (*optional*)
     Builds 'send-pr.info' from 'send-pr.texi' ('send-pr.info' is included with this distribution)
   - install-info (*optional*)
     Installs 'send-pr.info' into ' prefix/info'
   - clean (*optional*)
     Removes all intermediary build files that can be rebuilt from source code
4. Run, using the following command.

   ```
   install-sid your-sid
   ```

   your-sid is the identification code you received with send-pr. send-pr automatically inserts this value into the template field, >Submitter-Id:. If

[†] If your main Emacs lisp repository is in a different directory from this one, substitute that directory for '*prefix*/lib/emacs/lisp'.

you've downloaded send-pr from the Net, use 'net' for this value.

5. Place the following line in the path pointing to *prefix*/lib/emacs/lisp/default.el or instruct your users to place the following line in their '.emacs' files.

```
(autoload 'send-pr "send-pr" "Submit a Problem Report.")
```

6. Create a mail alias for the Support Site where you received send-pr, and for every site with which you wish to use send-pr to communicate.

   Each alias must have a suffix of '-prms'.

   The Support Site(s) will provide the correct addresses where these aliases should point. send-pr automatically searches for these aliases when you use the following input.

```
send-pr cygnus send-pr customername send-pr site...
```

   send-pr also uses *site* to determine the categories of problems accepted by the site in question by looking in a path something like the following location corresponds.

```
prefix/lib/prms/site
```

3

Processing `send-pr` problem reports

With each shipment of the GNUPro Toolkit, customers receive the latest version of `send-pr` with an up-to-date listing of valid categories (values for the **`>Category:`** field; see “Valid Categories” on page 67 for a complete list of the categories).

As an example, let’s pretend that you’re having a problem compiling some software and that you’re using the GNU C compiler that Cygnus supports. Assume that you’re getting an error in the `g++` program wherein the `prestidigitation` routines don’t match with the sources, `whatsitsname`.

You’ve made sure you’re following the rules of the program and checked and found that the bug isn’t already known (see the list of reported bugs in “Problems Fixed in this Release” in *Installation* in ***Getting Started with GNUPro Toolkit***).

In other words, you’re pretty sure you’ve found a bug and you’re going to report the bug using `send-pr`.

It really doesn’t matter whether you use `send-pr` from the shell or from within Emacs. If you use Emacs as a primary editor, calling `send-pr` from the shell is likely to start `send-pr` anyway in an Emacs buffer.

Since your company, Imaginary Software, Ltd., uses GNU software extensively, you’re pretty familiar with Emacs; so, from Emacs, in the buffer, use the keystroke combination, M-x (the diamond key called “Meta” simultaneously pressed with, in this case, the x key) and then type `send-pr` in the buffer.

You’ll then get a PR form (like the following example shows) to display on your

monitor.

You need to set certain information into each field, so you compile all the information you know about your problem. You have some sample code which you know should work, although it doesn't, so you'll include it.

```
SEND-PR: -*- text -*-
SEND-PR: Lines starting with 'SEND-PR' will be removed
SEND-PR: automatically as well as all comments (the text
SEND-PR: enclosed in '<' and '>').
SEND-PR: <Please consult the manual if you are not sure
SEND-PR: how to fill out a problem report.>
SEND-PR:
SEND-PR: Choose from the following categories:
SEND-PR:
SEND-PR: bfd        binutils    bison       byacc       config
SEND-PR: cvs        diff        doc         emacs       flexg++
SEND-PR: gas        gcc         gdb         globgprof   grep
SEND-PR: info       ispell      kerberos    ld          libg++
SEND-PR: libiberty  make        makeinfo    mas         newlib
SEND-PR: other      patch       rcs         readline    send-pr
SEND-PR: test       texindex    texinfo     texinfo.tex
SEND-PR:

To: cygnus-bugs@cygnus.com
Subject:
From: jeffrey@imaginary.com
Reply-To: jeffrey@imaginary.com
X-send-pr-version: send-pr 3.98-98r2

>Submitter-Id: imaginary
>Originator: Jeffrey Osier
>Organization: Imaginary Software, Ltd.
>Confidential: <[ yes | no ] (one line)>
>Synopsis: <synopsis of the problem (one line)>
>Severity: <[ non-critical | serious | critical ] (one line)>
>Priority: <[ low | medium | high ] (one line)>
>Category: <name of the product (one line)>
>Class: <[sw-bug|doc-bug|change-request|support](oneline)>
```

```
>Release: <release number or tag (oneline)>
>Environment: <machine, os, target, libraries (multiple lines)>

System: SunOS imaginary.com 4.1.1 1 sun4
Architecture: sun4

>Description: <precise description of the problem (multiple lines)>
>How-To-Repeat: <code/input/activities to reproduce (multiple lines)>
>Fix:
-----Emacs: *send-pr* (send-pr Fill)----All---------------------
>Category: other[]
```

What follows is a complete PR which you would send using the keystroke combination, C-c, C-c.

```
SEND-PR: Lines starting with 'SEND-PR' will be removed
SEND-PR: automatically as well as all comments.
SEND-PR: ...
SEND-PR:
To: cygnus-bugs@cygnus.com
Subject: g++ routines don't match
From: jeffrey@imaginary.com
Reply-To: jeffrey@imaginary.com
X-send-pr-version: send-pr 3.98-98r2

>Submitter-Id: imaginary
>Originator: Jeffrey Osier
>Organization: Imaginary Software, Ltd.
>Confidential: no
>Synopsis: g++ routines don't match
>Severity: serious
>Priority: medium
>Category: g++
>Class: sw-bug
>Release: progressive-98r2
>Environment:
System: SunOS imaginary.com 4.1.1 1 sun4
Architecture: sun4 (SPARC)
```

```
>Description:
   The following code I fed into the g++ came
   back with a strange error. apparently, the prestidigitation
   routine doesn't match with the whatsitsname in all cases.

>How-To-Repeat: call g++ using the following code.
                  ...code sample...
>Fix:
-----Emacs: *send-pr* (send-pr Fill)----All--------------------
To send the problem report use: C-c C-c
```

You use the keystroke combination, C-c, C-c, and off the report goes to Cygnus. Soon afterward, you get the reply that the bug has been accepted and forwarded to the responsible party.

```
From: prms (PRMS management)
Sender: prms-admin
Reply-To: hacker@cygnus.com
To: jeffrey@imaginary.com
Subject: Re: g++/1425: routines don't match

Thank you very much for your problem report.
It has the internal identification: g++/1425.
The individual assigned to look at your bug is: hacker
(F.B. Hacker)

Category: g++
Responsible: hacker
Synopsis: g++ routines don't match
Arrival-Date: Sat Feb 30 03:12:55 1997
```

A while later, you get a further analysis.

```
To: jeffrey@imaginary.com
From: hacker@cygnus.com
Subject: Re: g++/1425: routines don't match
Reply-To: hacker@cygnus.com

Got your message, Jeff. It seems that g++was confusing the
prestidigitation routines with the realitychecker when lexically
parsing the whatsitsname.
```

```
I'm working on robustisizing g++ now.

How about lunch next week?
--
F.B. Hacker
Cygnus, Sunnyvale, CA 408 542 9601
#include <std-disclaimer.h>
```

About the same time, you get another message, showing the problem has been analyzed and Cygnus is working on a solution.

```
From: hacker@cygnus.com
To: jeffrey@imaginary.com
Subject: Re: g++/1425: doesn't match prestidig
Reply-To: hacker@cygnus.com
            'F.B. Hacker' changed the state to 'analyzed'.

State-Changed-From-To: open-analyzed
State-Changed-By: hacker
State-Changed-When: Fri Feb 31 1997 08:59:16 1997
State-Changed-Why: figured out the problem, working on a patch
                     this afternoon
--
F.B. Hacker
Cygnus, Sunnyvale, CA 408 542 9601
#include <std-disclaimer.h>
```

Sometime later, you get more mail from F.B.:

```
To: jeffrey@imaginary.com
From: hacker@cygnus.com
Subject: Re: g++/1425: routines don't match
Reply-To: hacker@cygnus.com

There's a patch now that you can ftp over and check out.

F.B. Hacker
Cygnus, Sunnyvale, CA 408 542 9601
#include <std-disclaimer.h>
```

And then, you use ftp to get the fix for the bug. And again, you receive another status

message.

```
From: hacker@cygnus.com
To: jeffrey@imaginary.com
Subject: Re: g++/1425: doesn't match prestidig
Reply-To: hacker@cygnus.com

            'F.B. Hacker' changed the state to 'feedback'.

State-Changed-From-To: analyzed-feedback
State-Changed-By: hacker
State-Changed-When: Fri Feb 31 1997 23:43:16 1997
State-Changed-Why:
     got the patch finished, notified Jeff at Imaginary Software
--
F.B. Hacker
Cygnus, Sunnyvale, CA 408 542 9601
#include <std-disclaimer.h>
```

The bug has gone into `feedback` status now, until you get the patch, install it and test it. When everything tests well, you mail F.B. back and tell him the bug's been fixed, and he can change the state of the PR from `feedback` to `closed`.

Valid Categories

The following list describes valid entries for `>Category:`.

`bfd`
: GNU binary file descriptor library.

`binutils`
: GNU utilities for binary files (`ar`, `nm`, `size`...).

`bison`
: GNU parser generator.

`byacc`
: Free parser generator.

`config`
: Cygnus Software configuration and installation.

`cvs`
: Concurrent Version System.

`diff`
: GNU `diff` program.

`doc`
: Documentation and manuals.

`emacs`
: GNU Emacs editor and related functions.

`flex`
: GNU lexical analyzer.

`g++`
: GNU C++ compiler.

`gas`
: GNU assembler.

`gcc`
: GNU C compiler.

`gdb`
: GNU source code debugger.

`glob`
: The filename globbing functions.

`gprof`
: GNU profiler.

`grep`
: GNU `grep` program.

`info`
: GNU `info` hypertext reader.

`ispell`
: GNU spelling checker.

`kerberos`
: Kerberos authentication system.

`ld`
: GNU linker.

`libc`
: Cygnus C Library.

`libg++`
: GNU C++ class library.

`libiberty`
: GNU ‘`libiberty`’ library.

`libm`
: Cygnus Math Library.

`make`
: GNU `make` program.

`makeinfo`
: GNU utility to build Info files from Texinfo documents.

`mas`
: GNU Motorola syntax assembler.

`newlib`
: Cygnus C and Math Libraries.

`patch`
: GNU bug patch program.

`prms`
: GNU Problem Report Management System.

`rcs`
: Revision Control System.

`readline`
: GNU `readline` library.

`send-pr`
: GNU Problem Report submitting program.

`test`
: Category to use when testing `send-pr`.

`texindex`
: GNU documentation indexing utility.

`texinfo`
: GNU documentation macros.

`other`
: Anything which is not covered by the previous categories.

4

Details about `send-pr` and PRMS

A *Problem Report* is a message that describes a problem you are having with a body of work. `send-pr` organizes this message into a form which can be understood and automatically processed by PRMS, the GNU Problem Report Management System. A Problem Report is organized into fields which contain data describing you, your organization, and the problem you are announcing (see “Problem report format” on page 75). Problem Reports go through several defined states in their life-times, from open to closed (see the next discussion, “States of Problem Reports” on page 74).

States of Problem Reports

Each PR goes through a defined series of states between origination and closure. The originator of a PR receives notification automatically of any state changes.

`open`
: The initial state of a Problem Report where the PR has been filed and the responsible person(s) notified.

`analyzed`
: The responsible person has analyzed the problem, giving a preliminary evaluation of the problem and an estimate of the amount of time and resources necessary to solve the problem. It should also suggest possible workarounds.

`feedback`
: The problem has been solved, and the originator has been given a patch or other fix. The PR remains in this state until the originator acknowledges that the solution works.

`closed`
: A Problem Report is closed only when any changes have been integrated, documented, and tested, and the individual who submitted the problem report has confirmed the solution.

`suspended`
: Work on the problem is postponed. This happens if a timely solution is unlikely or is not cost-effective at the present time. The PR continues to exist, though a solution is not being actively sought. If the problem cannot be solved at all, it should be closed rather than suspended.

Problem report format

The format of a PR is designed to reflect the nature of PRMS as a database. Information is arranged into *fields*, and kept in individual records (PRs).

Problem Report fields are denoted by a keyword which begins with '>' and ends with ':', as in '`>Confidential:`'. Fields belong to one of the following three data types: ***Enumerated***, ***Text***, or ***MultiText***.

Enumerated

One of a specific set of values, varying according to the field. The value for each keyword must be on the same line as the keyword. These values are not configurable (yet).

For each ***Enumerated*** keyword, the possible choices are listed in the `send-pr` template as a comment. See the descriptions of fields for each field's explanations in detail.

The following fields are ***Enumerated*** format: `>Confidential:`, `>Severity:`, `>Priority:`, `>Class:`, `>State:` and `>Number:`.

Text

One single line of text which must begin and end on the same line (i.e., before a newline) as the keyword. See the descriptions of fields below for explanations of each field in detail.

The following fields are ***Text*** format: `>Submitter-Id:`, `>Originator:`, `>Synopsis:`, `>Category:`, `>Release:`, `>Responsible:` and `>Arrival-Date:`.

MultiText

Text of any length may occur in this field. ***MultiText*** may span multiple lines and may also include blank lines. A ***MultiText*** field ends only when another keyword appears. See the descriptions of fields for each field's explanations in detail.

The following fields are ***MultiText*** format: `>Organization:`, `>Environment:`, `>Description:`, `>How-To-Repeat:`, `>Fix:`, `>Audit-Trail:`, and `>Unformatted:`.

A Problem Report contains two different types of fields: *Mail Header* fields (see "Mail header fields" on page 77), used by the mail handler for delivery, and *Problem Report* fields (see "Problem report fields" on page 77), containing information relevant to the Problem Report and to the individual who submitted the problem's report. A Problem Report is essentially a specially formatted electronic mail message.

The following is an example Problem Report. Mail headers are at the top, followed by PRMS fields, which begin with '>' and end with ':'. The '`Subject:`' line in the mail header and the '`>Synopsis:`' field are usually duplicates of each other.

`Message-Id:` *message-identification*

```
Date: date
From: address
Reply-To: address
To: bug-address
Subject: subject
>Number: PRMS-id
>Category: category
>Synopsis: synopsis
>Confidential: yes or no
>Severity: critical, serious, or non-critical
>Priority: high, medium or low
>Responsible: responsible
>State: open, analyzed, suspended, feedback, or closed
>Class: sw-bug, doc-bug, change-request, support, or duplicate
>Submitter-Id: submitter-id
>Arrival-Date: date
>Originator: name
>Organization: organization
>Release: release
>Environment: environment
>Description: description
>How-To-Repeat: how-to-repeat
>Fix: fix
>Audit-Trail: appended-messages...
State-Changed-From-To: from- to
State-Changed-When: date
State-Changed-Why: reason
Responsible-Changed-From-To: from- to
Responsible-Changed-When: date
Responsible-Changed-Why: reason
>Unformatted: miscellaneous
```

See “Mail header fields” on page 77 for discussion of the mail headers at the top of problem reports. See “Problem report fields” on page 77 for discussion of the other fields and their contents.

Mail header fields

A Problem Report may contain any mail header field described in the Internet standard RFC-822. However, only the fields which identify the sender and the subject are required by `send-pr`.

`To:`
: The pre-configured mail address for the Support Site where the PR is to be sent, automatically supplied by `send-pr`.

`Subject:`
: A terse description of the problem. This field normally contains the same information as the '`>Synopsis:`' field.

`From:`
: Usually supplied automatically by the originator's mailer, containing the originator's electronic mail address.

`Reply-To:`
: A return address to which electronic replies can be sent; in most cases, the same address as the `From:` field.

Problem report fields

The other fields present whenever using `send-pr` are what the following discussions explain.

`>Submitter-Id:`
: (***Text***)
A unique identification code assigned by the Support Site. It is used to identify all Problem Reports coming from a particular site. (Submitters without a value for this field can invoke `send-pr` with the PRMS option, `--request-id`, to apply for one from the support organization. Problem reports from those not affiliated with the support organization should use the default value of '`net`' for this field.)

`>Originator:`
: (***Text***)
Originator's real name. The default is the value of the originator's environment variable, `NAME`.

`>Organization:`
: (***MultiText***)
The originator's organization. The default value is set with the variable, `DEFAULT_ORGANIZATION`, in the `send-pr` shell script.

`>Confidential:`
(***Enumerated***)
Use of this field depends on the originator's relationship with the support organization; contractual agreements often have provisions for preserving confidentiality. Conversely, a lack of a contract often means that any data provided will not be considered confidential. Submitters should contact the support organization directly if this is an issue. If the originator's relationship to the support organization provides for confidentiality, then, if the value of the field is '`yes`' the support organization treats the PR as confidential; any code samples provided are not made publicly available (such as in regression test suites). The default value is '`yes`'.

`>Synopsis:`
(***Text***)
One-line summary of the problem. send-pr copies this information to the '`Subject:`' line when you submit a Problem Report.

`>Severity:`
(***Enumerated***)
The severity of the problem. Accepted values include the following input.

`critical`
The product, component or concept is completely non-operational or some essential functionality is missing. No workaround is known.

`serious`
The product, component or concept is not working properly or significant functionality is missing. Problems that would otherwise be considered '`critical`' are rated '`serious`' when a workaround is known.

`non-critical`
The product, component or concept is working in general, but lacks features, has irritating behavior, does something wrong, or doesn't match its documentation.

The default value is '`serious`'.

`>Priority:`
(***Enumerated***)
How soon the originator requires a solution. Accepted values include the following input.

`high`
A solution is needed as soon as possible.

`medium`
The problem should be solved in the next release.

`low`
The problem should be solved in a future release.

The default value is ‘`medium`’.

`>Category:`
(***Text***)
The name of the product, component or concept where the problem lies. The values for this field are defined by the Support Site.

`>Class:`
(***Enumerated***)
The class of a problem uses one of the following subjects as input.

`sw-bug`
A general product problem. (‘`sw`’ stands for *software*.)

`doc-bug`
A problem with the documentation.

`change-request`
A request for a change in behavior, etc.

`support`
A support problem or question.

`duplicate (`*`pr-number`*`)`
Duplicate PR. *`pr-number`* should be the number of the original PR.

The default is ‘`sw-bug`’.

`>Release:`
(***Text***)
Release or version number of the product, component or concept.

`>Environment:`
(***MultiText***)
Description of the environment where the problem occurred: machine architecture, operating system, host and target types, libraries, pathnames, etc.

`>Description:`
(***MultiText***)
Precise description of the problem.

`>How-To-Repeat:`
(***MultiText***)
Example code, input, or activities to reproduce the problem. The support organization uses example code both to reproduce the problem and to test whether the problem is fixed. Include all preconditions, inputs, outputs, conditions after the problem, and symptoms. Any additional important information should be included. Include all the details that would be necessary for someone else to recreate the problem reported, however obvious. Sometimes seemingly arbitrary or obvious information can point the way toward a solution. See “Helpful hints” on page 92.

`>Fix:`
(***MultiText***)
A description of a solution to the problem, or a patch solving the problem. (This field is most often filled in at the Support Site; it is provided in case the submitter has solved the problem.)

PRMS adds the following fields when the PR arrives at the Support Site.

`>Number:`
(***Enumerated***)
The incremental identification number for this PR.

The '`>Number:`' field is often paired with the '`>Category:`' field in subsequent messages, as in the following input.

```
category/number
```

This is for historical reasons as well as because PRs are stored in sub-directories which are named by category.

`>State:`
(***Enumerated***)
The current state of the PR. Accepted values use the following descriptions. (The initial state of a PR is '`open`'; see "States of Problem Reports" on page 74 for more discussion.)

`open`
The PR has been filed and the responsible person notified.

`analyzed`
The responsible person has analyzed the problem.

`feedback`
The problem has been solved, and the originator has been given a patch or other fix.

`closed`
The changes have been integrated, documented, and tested, and the originator has confirmed that the solution works.

`suspended`
Work on the problem has been postponed.

`>Responsible:`
(***Text***)
The person responsible for this category.

`>Arrival-Date:`
(***Text***)
The time that this PR was received by PRMS. The date is provided automatically by PRMS.

`>Audit-Trail:`
(***MultiText***)
Tracks related electronic mail as well as changes in the '`>State:`' and '`>Responsible:`' fields with the following sub-fields:

`State-Changed-<From>-<To>: oldstate>-< newstate`
The old and new '`>State:`' field values.

`Responsible-Changed-<From>-<To>: oldresp>-< newresp`
The old and new '`>Responsible:`' field values.

`State-Changed-By: name`

`Responsible-Changed-By: name`
The name of the maintainer who effected the change.

`State-Changed-When: timestamp`

`Responsible-Changed-When: timestamp`
The time the change was made.

`State-Changed-Why: reason`...

`Responsible-Changed-Why: reason`...
The reason for the change.

The '`>Audit-Trail:`' field also contains any mail messages received by PRMS related to the submitted PR, in the order in which the electronic mail was received.

`>Unformatted:`
(***MultiText***)
Any random text found outside the fields in the original Problem Report.

5

Editing and sending PRs

You can invoke `send-pr` from a shell prompt or from within GNU Emacs using the keystroke combination, M-x (using the diamond-shaped 'Meta' key simultaneously with the 'x' key), and then typing `send-pr` in the buffer.

Creating new Problem Reports

Invoking `send-pr` presents a PR template with a number of fields already filled in. Complete the template as thoroughly as possible to make a useful bug report. Submit only one bug with each PR. A template consists of the following three sections.

- ***Comments***
  The top several lines of a blank template consist of a series of comments that provide some basic instructions for completing the Problem Report, as well as a list of valid entries for the '**`>Category:`**' field. These comments are all preceded by the '**`SEND-PR:`**' string and are erased automatically when the PR is submitted. The instructional comments within '**`<`**' and '**`>`**' are also removed. (Only these comments are removed; lines you provide that happen to have those characters in them, such as examples of shell-level redirection, are not affected.)
- ***Mail Header***
  send-pr creates a standard mail header. send-pr completes all fields except the '`Subject:`' line with default values. See "Problem report format" on page 75.
- ***PRMS fields***
  These are the informational fields that PRMS uses to route your Problem Report to the responsible party for further action. They should be filled out as completely as possible. (See "Processing send-pr problem reports" on page 61, "Problem report format" on page 75, and "Helpful hints" on page 92.)

The default template contains your pre-configured '**`>Submitter-Id:`**'.

`send-pr` attempts to determine values for the '**`>Originator:`**' and '**`>Organization:`**' fields (see "Problem report format" on page 75). `send-pr` will set the '**`>Originator:`**' field to the value of the `NAME` environment variable if it has been set; similarly, '**`>Organization:`**' will be set to the value of `ORGANIZATION`. `send-pr` also attempts to find out some information about your system and architecture, and, in the '**`>Environment:`**' field, places it if found.

You may submit problem reports to different Support Sites from the default site by specifying the alternate site when you invoke `send-pr`. Each `site` has its own list of categories for which it accepts Problem Reports. (See "Setting a default site" on page 58.)

`send-pr` also provides the mail header section of the template with default values in the '**`To:`**', '**`From:`**', and '**`Reply-To:`**' fields. The '**`Subject:`**' field is empty.

The template begins with a comment section like the following example.

```
SEND-PR: -*- send-pr -*-
SEND-PR: Lines starting with 'SEND-PR' will be removed
```

```
SEND-PR: automatically as well as all comments (the text
SEND-PR: below enclosed in '<' and '>').
SEND-PR:
SEND-PR: Please consult the document 'Reporting Problems' if
SEND-PR: you are not sure how to fill out a problem report.
SEND-PR: Choose from the following categories:
```

The template also contains a list of valid **`>Category:`** values for the Support Site to whom you are submitting this Problem Report. One (and only one) of these values should be placed in the **`>Category:`** field. For an example of a complete sample bug report (from template to completed PR), see "Processing send-pr problem reports" on page 61. For a complete list of valid categories, type '`send-pr -L`' at your prompt. See "Valid Categories" on page 67 for a sample list of categories.

The mail header is just below the comment section. Fill out the '**`Subject:`**' field, if it is not already completed using the value of '**`>Synopsis:`**'. The other mail header fields contain default values.

```
To: support-site
Subject: complete this field
From: your-login@your-site
Reply-To: your-login@your-site
X-send-pr-version: send-pr version
```

`support-site` in the '**`To:`**' field is an alias for the Support Site to which you wish to submit this PR.

The rest of the template contains PRMS fields. Each field is either automatically completed with valid information (such as your '**`>Submitter-Id:`**') or contains a one-line instruction specifying the information that field requires in order to be correct. For example, the '**`>Confidential:`**' field expects a value of '`yes`' or '`no`', and the answer must fit on one line; similarly, the '**`>Synopsis:`**' field expects a short synopsis of the problem, which must also fit on one line. Fill out the fields as completely as possible. See "Helpful hints" on page 92 for suggestions as to what kinds of information to include.

In this example, words in italics are filled in with pre-configured information, asin the following example report.

```
>Submitter-Id: submitter's identification
>Originator: submitter's name here
>Organization: submitter's organization
>Confidential:<[ yes | no ] (one line)>
>Synopsis: <synopsis of the problem (one line)>
>Severity: <[non-critical | serious | critical] (one line)>
>Priority: <[ low | medium | high ] (one line)>
>Category: <name of the product (one line)>
>Class: <[sw-bug | doc-bug | change-request | support]>
>Release: <version (one line)>
```

```
>Environment:  <machine, operating system, target, libraries>
                  (multiple lines)
>Description: <precise description of the problem>
                  (multiple lines)
>How-To-Repeat: <code/input/activities to reproduce>
                  (multiple lines)
>Fix: <how to correct or work around the problem, if known>
                  (multiple lines)
```

When you finish editing the Problem Report, `send-pr` mails it to the address named in the '**`To:`**' field in the mail header. `send-pr` checks that the complete form contains a valid '**`>Category:`**'.

Your copy of `send-pr` should have already been customized on installation to reflect your '**`>Submitter-Id:`**'. See "Installing send-pr on your system" on page 57. If you don't know your '**`>Submitter-Id:`**', you can request it using '`send-pr --request-id`'. If your organization is not affiliated with the site you send Problem Reports to, a good generic '**`>Submitter-Id:`**' to use is '`net`'.

If your PR has an invalid value in one of the Enumerated fields (see "Problem report format" on page 75), `send-pr` places the PR in a temporary file named '`/tmp/pbad`*nnnn*' on your machine. *nnnn* is the process identification number given to your current `send-pr` session. If you are running `send-pr` from the shell, you are prompted as to whether or not you wish to try editing the same Problem Report again. If you are running `send-pr` from Emacs, the Problem Report is placed in the buffer '`*send-pr-error*`'; you can edit this file and then submit it using the keystroke combination, `M-x`, and then typing `prms-submit-pr` in the buffer. Any further mail concerning this Problem Report should be `cc:`'d to the PRMS mailing address as well, with the category and identification number in the '**`Subject:`**' line of the message like the following example shows.

`Subject:` `Re:` *`pr category/prms-id`*`:` *`original message subject`*

Messages arriving with '**`Subject:`**' lines of this form are automatically appended to the report in the '**`>Audit-Trail:`**' field in the order received.

Using send-pr from within Emacs

You can use an interactive send-pr interface from within GNU Emacs to fill out your Problem Report. We recommend that you familiarize yourself with Emacs before using this feature (see “Introduction” in ***The GNU Emacs Manual***).

Call send-pr with the keystroke combination, M-x, along with typing send-pr[†] in the buffer.

send-pr responds with a Problem Report template pre-configured for the Support Site from which you received send-pr.

If you use send-pr locally, the default Support Site is probably your local site.

You may also submit problem reports to different Support Sites from the default site. To use this feature, invoke send-pr with the keystroke combination, C-u, M-x, along with typing send-pr in the buffer.

send-pr prompts you for the name of a *site*. *site* is an alias on your local machine which points to an alternate Support Site.

send-pr displays the template and prompts you in the minibuffer with the following line.

```
>Category: other
```

Delete the default value ‘other’ in the minibuffer and replace it with the keyword corresponding to your problem (the list of valid categories is in the topmost section of the PR template). For example, if the problem you wish to report has to do with the GNU C compiler, and your support organization accepts bugs submitted for this program under the category ‘gcc’, delete ‘other’ and then type gcc and press Enter. send-pr replaces the following line.

```
>Category: <name of the product (one line)>
```

The previous example’s input is replaced in the template with the following, and send-pr moves on to another field.

```
>Category: gcc
```

send-pr provides name completion in the minibuffer. For instance, you can also type gc and use the TAB key, and send-pr attempts to complete the entry for you. Typing g and using the TAB key may not have the same effect if several possible entries begin with ‘g’. In that case send-pr cannot complete the entry because it cannot determine whether you mean ‘gcc’ or, for instance, ‘gdb’, since both of those are possible categories. send-pr continues to prompt you for a valid entry until you enter one.

[†] If using the keystroke combination, M-x, and then typing send-pr doesn’t work, see your system administrator for loading send-pr.

send-pr prompts you interactively to enter each field for which there is a range of specific choices. If you attempt to enter a value which is not in the range of acceptable entries, send-pr responds with the message, [No match], and allows you to change the entry until it contains an acceptable value. This avoids unusable information (at least in these fields) and also avoids typographical errors which could cause problems later.

send-pr prompts you for the following fields.

```
>Category:
>Confidential: (default: no)
>Severity: (default: serious)
>Priority: (default: medium)
>Class: (default: sw-bug)
>Release:
>Synopsis: (this value is copied to Subject:)
```

After you complete these fields, send-pr places the cursor in the '**>Description:**' field and displays the following message in the minibuffer.

```
To send the problem report use: C-c C-c
```

At this point, edit the file in the main buffer to reflect your specific problem, putting relevant information in the proper fields. See "Processing send-pr problem reports" on page 61 for a sample problem report. send-pr provides the following keystroke combinations, called *bindings*, to make moving around in a template buffer more simple.

C-c, C-f
M-x change-field
: Changes the field under the cursor. edit-pr prompts you then for a new value.

M-C, b
M-x, prms-backward-field
: Moves the cursor to the beginning of the value of the current field.

M-C, f
M-x prms-forward-field
: Moves the cursor to the end of the value of the current field.

M-p
M-x prms-previous-field
: Moves the cursor back one field to the beginning of the value of the previous field.

M-n
M-x prms-next-field
: Moves the cursor forward one field to the beginning of the value of the next field.

send-pr takes over again when you use the keystroke combination, C-c C-c, and sends the message. send-pr reports any errors in a separate buffer, which remains in

existence until you send the PR properly (or, of course, until you explicitly kill the buffer). For detailed instructions on using Emacs, see “Introduction” in ***The GNU Emacs Manual***.

Invoking send-pr from the shell

```
send-pr  [ site ]
          [ -f problem-report | --file problem-report ]
          [ -t mail-address | --to mail-address ]
          [ --request-id ]
          [ -L | --list ] [ -P | --print ]
          [ -V | --version] [ -h | --help ]
```

site is an alias on your local machine which points to an address used by a Support Site. If this argument is not present, the default *site* is usually the site which you received send-pr from, or your local site if you use PRMS locally. See “Setting a default site” on page 58.

Invoking send-pr with no options calls the editor named in your environment variable, EDITOR, on a default PR template.

If the environment variable, PR_FORM, is set, its value is used as a file name which contains a valid template.

If PR_FORM points to a missing or unreadable file, or if the file is empty, send-pr generates an error message and opens the editor on a default template.

The following commands and arguments are what to use for PRs.

`-f` *problem-report*
`--file` *problem-report*
: Specifies a file, *problem-report*, where a completed Problem Report already exists. send-pr sends the contents of the file without invoking an editor.

  If *problem-report* is ‘-’, send-pr reads from standard input.

`-t` *mail-address*
`--to` *mail-address*
: Sends the PR to *mail-address*.

  The default is preset when send-pr is configured.

  This option is not recommended; instead, use the argument, *site*, on the command line.

`-c` *mail-address*
`--cc` *mail-address*
: Places *mail-address* in the **Cc:** header field of the message to be sent.

`--request-id`
: Sends a request for a **>Submitter-Id:** to the Support Site.

`-L`
`—list`
: Prints the list of valid >Category: values on standard output. No mail is sent.

`-s` *severity*
`--severity` *severity*
: Sets the initial value of the **`>Severity:`** field to severity (meaning the *state* of severity).

`-P`
`--print`
: Displays the PR template.

  If the variable, `PR_FORM`, is set in your environment, the file it specifies is printed.

  If `PR_FORM` is not set, `send-pr` prints the standard blank form.

  If the file specified by `PR_FORM` doesn't exist, `send-pr` displays an error message. No mail is sent.

`-V`
`--version`
: Displays the `send-pr` version number and a usage summary. No mail is sent.

`-h`
`--help`
: Displays a usage summary for `send-pr`. No mail is sent.

Helpful hints

There is no orthodox standard for submitting effective bug reports, though you might do well to consult the section on submitting bugs for GNU `gcc` in "Reporting Bugs" in ***Using GNU CC***.

The following contains instructions on what kinds of information to include and what kinds of mistakes to avoid.

In general, common sense (assuming such an animal exists) dictates the kind of information that would be most helpful in tracking down and resolving problems in software.

- Include anything *you* would want to know if you were looking at the report from the other end. There's no need to include every minute detail about your environment, although anything that might be different from someone else's environment should be included (your path, for instance).
- Narratives are often useful, given a certain degree of restraint. If a person responsible for a bug can see that A was executed, and then B and then C, knowing that sequence of events might trigger the realization of an intermediate step that was missing, or an extra step that might have changed the environment enough to cause a visible problem. Again, restraint is always in order (comments such as "I set the build running, went to get a cup of coffee—Columbian, cream, no sugar—talked to Sheila on the phone, and then *THIS* happened..." is not exemplary of restraint) and be sure to include anything relevant.
- Richard Stallman writes, "The fundamental principle of reporting bugs usefully is this: ***report all the facts***. If you are not sure whether to state a fact or leave it out, state it!" This holds true across all problem reporting systems, for computer software or social injustice or motorcycle maintenance. It is especially important in the software field due to the major differences seemingly insignificant changes can make (a changed variable, a missing semicolon, etc.).
- Submit only one problem with each Problem Report. If you have multiple problems, use multiple PRs. This aids in tracking each problem and also in analyzing the problems associated with a given program.
- It never hurts to do a little research to find out if the bug you've found has already been reported. See "Problems Fixed in this Release" in *Installation* in ***Getting Started with GNUPro Toolkit*** for a list of known bugs which come with the software; see your system administrator if you don't have a copy of this documentation.

- The more closely a PR adheres to the standard format, the less interaction is required by a database administrator to route the information to the proper place. Keep in mind that anything that requires human interaction also requires time that might be better spent in actually fixing the problem. It is therefore in everyone's best interest that the information contained in a PR be as correct as possible (in both format and content) at the time of submission.

GNPRO TOOLKIT™

Comparing & Merging Differences

98r1
July, 1998

CYGNUS

Copyright © 1988-1998 Free Software Foundation

Permission is granted to make and distribute verbatim copies of this documentation provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the "GNU General Public License" are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

This documentation has been prepared by Cygnus.

All rights reserved.

GNUPro™, the GNUPro™ logo, and the Cygnus logo are trademarks of Cygnus. All other brand and product names are trademarks of their respective owners.

To contact the Cygnus Technical Publications staff, email: `doc@cygnus.com`.

GNU `diff` was written by Mike Haertel, David Hayes, Richard Stallman, Len Tower, and Paul Eggert. Wayne Davison designed and implemented the unified output format.

The basic algorithm is described in "An O(ND) Difference Algorithm and its Variations" by Eugene W. Myers, in ***Algorithmica***; Vol. 1, No. 2, 1986; pp. 251–266; and in "A File Comparison Program" by Webb Miller and Eugene W. Myers, in ***Software—Practice and Experience***; Vol. 15, No. 11, 1985; pp. 1025–1040.

The algorithm was independently discovered as described in "Algorithms for Approximate String Matching" by E. Ukkonen, in ***Information and Control***; Vol. 64, 1985, pp. 100–118.

GNU `diff3` was written by Randy Smith.

GNU `sdiff` was written by Thomas Lord.

GNU `cmp` was written by Torbjorn Granlund and David MacKenzie.

`patch` was written mainly by Larry Wall; the GNU enhancements were written mainly by Wayne Davison and David MacKenzie. Parts of the documentation are adapted from a material written by Larry Wall, with his permission.

1

Overview of `diff` and `patch`

Computer users often find occasion to ask how two files differ. Perhaps one file is a newer version of the other file. Or maybe the two files initially were identical copies that were then changed by different people.

You can use the `diff` command to show differences between two files, or each corresponding file in two directories. `diff` outputs differences between files line by line in any of several formats, selectable by command line options. This set of differences is often called a `diff` or `patch`.

The following documentation discusses using the commands and other related commands.

For files that are identical, `diff` normally produces no output; for binary (non-text) files, `diff` normally reports only that they are different.

You can use the `cmp` command to show the offsets and line numbers where two files differ. `cmp` can also show all the characters that differ between the two files, side by side. Another way to compare two files character by character is the Emacs command, `M-x compare-windows`. See “Comparing Files” in ***The GNU Emacs Manual*** for more information, particularly on that command.

You can use the `diff3` command to show differences among three files. When two people have made independent changes to a common original, `diff3` can report the differences between the original and the two changed versions, and can produce a merged file that contains both persons’ changes together with warnings about conflicts.

You can use the `sdiff` command to merge two files interactively.

You can use the set of differences produced by `diff` to distribute updates to text files (such as program source code) to other people. This method is especially useful when the differences are small compared to the complete files. Given `diff` output, you can use the patch program to update, or patch, a copy of the file. If you think of `diff` as subtracting one file from another to produce their difference, you can think of patch as adding the difference to one file to reproduce the other.

This documentation first concentrates on making `diffs`, and later shows how to use `diffs` to update files.

2

What comparison means

There are several ways to think about the differences between two files. One way to think of the differences is as a series of lines that were deleted from, inserted in, or changed in one file to produce another file. `diff` compares two files line by line, finds groups of lines that differ, and reports each group of differing lines. It can report the differing lines in several formats, which have different purposes. See the following documentation for more information.

GNU `diff` can show whether files are different without detailing the differences. It also provides ways to suppress certain kinds of differences that are not important to you. Most commonly, such differences are changes in the amount of white space

between words or lines. `diff` also provides ways to suppress differences in alphabetic case or in lines that match a regular expression that you provide. These options can accumulate; for example, you can ignore changes in both white space and alphabetic case.

Another way to think of the differences between two files is as a sequence of pairs of characters that can be either identical or different. `cmp` reports the differences between two files character by character, instead of line by line. As a result, it is more useful than `diff` for comparing binary files. For text files, `cmp` is useful mainly when you want to know only whether two files are identical.

To illustrate the effect that considering changes character by character can have compared with considering them line by line, think of what happens if a single newline character is added to the beginning of a file. If that file is then compared with an otherwise identical file that lacks the newline at the beginning, `diff` will report that a blank line has been added to the file, while `cmp` will report that almost every character of the two files differs.

`diff3` normally compares three input files line by line, finds groups of lines that differ, and reports each group of differing lines. Its output is designed to make it easy to inspect two different sets of changes to the same file.

Hunks

When comparing two files, `diff` finds sequences of lines common to both files, interspersed with groups of differing lines called *hunks*. Comparing two identical files yields one sequence of common lines and no hunks, because no lines differ. Comparing two entirely different files yields no common lines and one large hunk that contains all lines of both files. In general, there are many ways to match up lines between two given files. `diff` tries to minimize the total hunk size by finding large sequences of common lines interspersed with small hunks of differing lines. For example, suppose the file '`F`' contains the three lines '`a`', '`b`', '`c`', and the file '`G`' contains the same three lines in reverse order '`c`', '`b`', '`a`'. If `diff` finds the line '`c`' as common, then the command '`diff F G`' produces the following output:

```
1,2d0
< a
< b
3a2,3
> b
> a
```

But if `diff` notices the common line '`b`' instead, it produces the following output:

```
1c1
< a
---
> c
3c3
< c
---
> a
```

It is also possible to find '`a`' as the common line. `diff` does not always find an optimal matching between the files; it takes shortcuts to run faster. But its output is usually close to the shortest possible. You can adjust this tradeoff with the '`--minimal`' option (see "diff performance tradeoffs" on page 137).

Suppressing differences in blank and tab spacing

The ‘`-b`’ and ‘`--ignore-space-change`’options ignore white space at line end, and considers all other sequences of one or more white space characters to be equivalent. With these options, `diff` considers the following two lines to be equivalent, where ‘**`$`**’ denotes the line end:

```
Here lyeth  muche rychnesse in lytell space. -- John Heywood$
Here lyeth muche rychnesse in lytell space. -- John Heywood $
```

The ‘`-w`’ and ‘`--ignore-all-space`’ options are stronger than ‘`-b`’. They ignore difference even if one file has white space where the other file has none. *White space* characters include tab, newline, vertical tab, form feed, carriage return, and space; some locales may define additional characters to be white space. With these options, `diff` considers the following two lines to be equivalent, where ‘**`$`**’ denotes the line end and ‘**`^M`**’ denotes a carriage return:

```
Here lyeth muche rychnesse in lytell space. --   John Heywood$
He relyeth much erychnes seinly tells pace. --John Heywood ^M$
```

Suppressing differences in blank lines

The ‘`-B`’ and ‘`--ignore-blank-lines`’ options ignore insertions or deletions of blank lines. These options normally affect only lines that are completely empty; they do not affect lines that look empty but contain space or tab characters. With these options, for instance, consider a file containing only the following.

```
1. A point is that which has no part.

2. A line is breadthless length.
-- Euclid, The Elements, I
```

The last example is considered identical to another file containing only the following.

```
1. A point is that which has no part.
2. A line is breadthless length.

-- Euclid, The Elements, I
```

Suppressing case differences

GNU `diff` can treat lowercase letters as equivalent to their uppercase counterparts, so that, for example, it considers ‘`Funky Stuff`’, ‘`funky STUFF`’, and ‘`fUNKy stuFf`’ to all be the same. To request this, use the ‘`-i`’ or ‘`--ignore-case`’ option.

Suppressing lines matching a regular expression

To ignore insertions and deletions of lines that match a regular expression, use the '`-I` *`regexp`*' or '`--ignore-matching-lines=` *`regexp`*' option. You should escape regular expressions that contain shell meta-characters to prevent the shell from expanding them.

For example, '`diff -I '^[0-9]'`' ignores all changes to lines beginning with a digit.

However, '`-I`' only ignores the insertion or deletion of lines that contain the regular expression if every changed line in the hunk—every insertion and every deletion—matches the regular expression.

In other words, for each non-ignorable change, `diff` prints the complete set of changes in its vicinity, including the ignorable ones.

You can specify more than one regular expression for lines to ignore by using more than one '`-I`' option. `diff` tries to match each line against each regular expression, starting with the last one given.

Summarizing which files differ

When you only want to find out whether files are different, and you don’t care what the differences are, you can use the summary output format. In this format, instead of showing the differences between the files, `diff` simply reports whether files differ. The ‘`-q`’ and ‘`--brief`’ options select this output format.

This format is especially useful when comparing the contents of two directories. It is also much faster than doing the normal line by line comparisons, because `diff` can stop analyzing the files as soon as it knows that there are any differences.

You can also get a brief indication of whether two files differ by using `cmp`. For files that are identical, `cmp` produces no output. When the files differ, by default, `cmp` outputs the byte offset and line number where the first difference occurs. You can use the ‘`-s`’ option to suppress that information, so that `cmp` produces no output and reports whether the files differ using only its exit status (see “Invoking cmp” on page 169).

Unlike `diff`, `cmp` cannot compare directories; it can only compare two files.

Binary files and forcing text comparisons

If `diff` thinks that either of the two files it is comparing is binary (a non-text file), it normally treats that pair of files much as if the summary output format had been selected (see “Summarizing which files differ” on page 106), and reports only that the binary files are different. This is because line by line comparisons are usually not meaningful for binary files.

`diff` determines whether a file is text or binary by checking the first few bytes in the file; the exact number of bytes is system dependent, but it is typically several thousand. If every character in that part of the file is non-null, `diff` considers the file to be text; otherwise it considers the file to be binary.

Sometimes you might want to force `diff` to consider files to be text. For example, you might be comparing text files that contain null characters; `diff` would erroneously decide that those are non-text files. Or you might be comparing documents that are in a format used by a word processing system that uses null characters to indicate special formatting. You can force `diff` to consider all files to be text files, and compare them line by line, by using the ‘`-a`’ or ‘`--text`’ option. If the files you compare using this option do not in fact contain text, they will probably contain few newline characters, and the `diff` output will consist of hunks showing differences between long lines of whatever characters the files contain.

You can also force `diff` to consider all files to be binary files, and report only whether they differ (but not how). Use the ‘`--brief`’ option for this.

In operating systems that distinguish between text and binary files, `diff` normally reads and writes all data as text. Use the ‘`--binary`’ option to force `diff` to read and write binary data instead. This option has no effect on a Posix-compliant system like GNU or traditional Unix. However, many personal computer operating systems represent the end of a line with a carriage return followed by a newline. On such systems, `diff` normally ignores these carriage returns on input and generates them at the end of each output line, but with the ‘`--binary`’ option `diff` treats each carriage return as just another input character, and does not generate a carriage return at the end of each output line. This can be useful when dealing with non-text files that are meant to be interchanged with Posix-compliant systems.

If you want to compare two files byte by byte, you can use the `cmp` program with the ‘`-l`’ option to show the values of each differing byte in the two files. With GNU `cmp`, you can also use the ‘`-c`’ option to show the ASCII representation of those bytes. See “Invoking cmp” on page 169 for more information.

If `diff3` thinks that any of the files it is comparing is binary (a non-text file), it normally reports an error, because such comparisons are usually not useful. `diff3`

uses the same test as `diff` to decide whether a file is binary. As with `diff`, if the input files contain a few non-text characters but otherwise are like text files, you can force `diff3` to consider all files to be text files and compare them line by line by using the '`-a`' or '`--text`' options.

3

`diff` output formats

`diff` has several mutually exclusive options for output format. The following documentation discusses the output and formats.

Two sample input files

The following are two sample files that we will use in numerous examples to illustrate the output of `diff` and how various options can change it. The following is the file, '**lao**'.

```
The Way that can be told of is not the eternal Way;
The name that can be named is not the eternal name.
The Nameless is the origin of Heaven and Earth;
The Named is the mother of all things.
Therefore let there always be non-being,
  so we may see their subtlety,
And let there always be being,
  so we may see their outcome.
The two are the same,
But after they are produced,
  they have different names.
```

The following is the file, '**tzu**'.

```
The Nameless is the origin of Heaven and Earth;
The named is the mother of all things.

Therefore let there always be non-being, so we may see their
  subtlety,
And let there always be being,
  so we may see their outcome.
The two are the same,
But after they are produced,
  they have different names.
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
```

In this example, the first hunk contains just the first two lines of '**lao**', the second hunk contains the fourth line of '**lao**' opposing the second and third lines of '**tzu**', and the last hunk contains just the last three lines of '**tzu**'.

Showing differences without context

The *normal* `diff` output format shows each hunk of differences without any surrounding context. Sometimes such output is the clearest way to see how lines have changed, without the clutter of nearby unchanged lines (although you can get similar results with the context or unified formats by using 0 lines of context). However, this format is no longer widely used for sending out patches; for that purpose, the context format (see “Context format” on page 113) and the unified format (see “Unified format” on page 115) are superior. Normal format is the default for compatibility with older versions of `diff` and the Posix standard.

Detailed description of normal format

The normal output format consists of one or more hunks of differences; each hunk shows one area where the files differ. Normal format hunks look like the following:

```
change-command
< from-file-line
< from-file-line ...
---
> to-file-line
> to-file-line ...
```

There are three types of change commands. Each consists of a line number or comma-separated range of lines in the first file, a single character indicating the kind of change to make, and a line number or comma-separated range of lines in the second file. All line numbers are the original line numbers in each file. The types of change commands are the following.

*l*a*r*
: Add the lines in range *r* of the second file after line *l* of the first file. For example, ‘`8a12,15`’ means append lines 12–15 of file 2 after line 8 of file 1; or, if changing file 2 into file 1, delete lines 12–15 of file 2.

*f*c*t*
: Replace the lines in range *f* of the first file with lines in range *t* of the second file. This is like a combined add and delete, but more compact. For example, ‘`5,7c8,10`’ means change lines 5–7 of file 1 to read as lines 8–10 of file 2; or, if changing file 2 into file 1, change lines 8–10 of file 2 to read as lines 5–7 of file 1.

*r*d*l*
: Delete the lines in range *r* from the first file; line *l* is where they would have appeared in the second file had they not been deleted. For example, ‘`5,7d3`’ means delete lines 5–7 of file 1; or, if changing file 2 into file 1, append lines 5–7 of file 1 after line 3 of file 2.

An example of normal format

The following is the output of the command '**`diff lao tzu`**' (see "Two sample input files" on page 110 for the complete contents of the two files).

Notice that the following example shows only the lines that are different between the two files.

```
1,2d0
< The Way that can be told of is not the eternal Way;
< The name that can be named is not the eternal name.
4c2,3
< The Named is the mother of all things.
---
> The named is the mother of all things.
>
11a11,13
> They both may be called deep and profound.
> Deeper and more profound,
> The door of all subtleties!
```

Showing differences in their context

Usually, when you are looking at the differences between files, you will also want to see the parts of the files near the lines that differ, to help you understand exactly what has changed. These nearby parts of the files are called the *context*.

GNU `diff` provides two output formats that show context around the differing lines: *context format* and *unified format*. It can optionally show in which function or section of the file the differing lines are found.

If you are distributing new versions of files to other people in the form of `diff` output, you should use one of the output formats that show context so that they can apply the diffs even if they have made small changes of their own to the files. `patch` can apply the diffs in this case by searching in the files for the lines of context around the differing lines; if those lines are actually a few lines away from where the `diff` says they are, `patch` can adjust the line numbers accordingly and still apply the `diff` correctly. See “Applying imperfect patches” on page 160 for more information on using `patch` to apply imperfect diffs.

Context format

The context output format shows several lines of context around the lines that differ. It is the standard format for distributing updates to source code.

To select this output format, use the ‘`-C` *`lines`*’, ‘`--context[=`*`lines`*`]`’, or ‘`-c`’ option. The argument *`lines`* that some of these options take is the number of lines of context to show. If you do not specify lines,it defaults to three. For proper operation, `patch` typically needs at least two lines of context.

Detailed description of context format

The context output format starts with a two-line header, which looks like the following.

```
*** from-file from-file-modification-time
--- to-file to-file-modification time
```

You can change the header’s content with the ‘`-L` *`label`*’ or ‘`--label=`*`label`*’ option; see “Showing alternate file names” on page 117. Next come one or more hunks of differences; each hunk shows one area where the files differ. Context format hunks look like the following.

```
***************
*** from-file-line-range ****
  from-file-line
  from-file-line ...
--- to-file-line-range ----
```

```
  to-file-line
  to-file-line...
```

The lines of context around the lines that differ start with two space characters. The lines that differ between the two files start with one of the following indicator characters, followed by a space character:

‘**!**’
: A line that is part of a group of one or more lines that changed between the two files. There is a corresponding group of lines marked with ‘!’ in the part of this hunk for the other file.

‘**+**’
: An “inserted” line in the second file that corresponds to nothing in the first file.

‘**-**’
: A “deleted” line in the first file that corresponds to nothing in the second file.

If all of the changes in a hunk are insertions, the lines of *`from-file`* are omitted. If all of the changes are deletions, the lines of *`to-file`* are omitted.

An example of context format

Here is the output of ‘`diff -c lao tzu`’ (see “Two sample input files” on page 110 for the complete contents of the two files). Notice that up to three lines that are not different are shown around each line that is different; they are the context lines. Also notice that the first two hunks have run together, because their contents overlap.

```
*** lao Sat Jan 26 23:30:39 1991
--- tzu Sat Jan 26 23:30:50 1991
***************
*** 1,7 ****
- The Way that can be told of is not the eternal Way;
- The name that can be named is not the eternal name.
  The Nameless is the origin of Heaven and Earth;
! The Named is the mother of all things.
  Therefore let there always be non-being,
    so we may see their subtlety,
  And let there always be being,
--- 1,6 ----
  The Nameless is the origin of Heaven and Earth;
! The named is the mother of all things.
!
  Therefore let there always be non-being,
    so we may see their subtlety,
  And let there always be being,
***************
*** 9,11 ****
--- 8,13 ----
  The two are the same,
```

```
  But after they are produced,
    they have different names.
+ They both may be called deep and profound.
+ Deeper and more profound,
+ The door of all subtleties!
```

An example of context format with less context

The following example shows the output of '`diff --context=1 lao tzu`' (see "Two sample input files" on page 110 for the complete contents of the two files). Notice that at most one context line is reported here.

```
*** lao Sat Jan 26 23:30:39 1991
--- tzu Sat Jan 26 23:30:50 1991
***************
*** 1,5 ****
- The Way that can be told of is not the eternal Way;
- The name that can be named is not the eternal name.
  The Nameless is the origin of Heaven and Earth;
! The Named is the mother of all things.
  Therefore let there always be non-being,
--- 1,4 ----
  The Nameless is the origin of Heaven and Earth;
! The named is the mother of all things.
!
  Therefore let there always be non-being,
***************
*** 11 ****
--- 10,13 ----
    they have different names.
+ They both may be called deep and profound.
+ Deeper and more profound,
+ The door of all subtleties!
```

Unified format

The unified output format is a variation on the context format that is more compact because it omits redundant context lines. To select this output format, use the '`-U` *`lines`*', '`--unified[=`*`lines`*`]`', or '`-u`' option. The argument linesis the number of lines of context to show. When it is not given, it defaults to three. At present, only GNU diff can produce this format and only GNU patch can automatically apply diffs in this format. For proper operation, patch typically needs at least two lines of context.

Detailed description of unified format

The unified output format starts with a two-line header, which looks like this:

```
--- from-file from-file-modification-time
+++ to-file to-file-modification-time
```

You can change the header's content with the '`-L label`' or '`--label=label`' option; see See "Showing alternate file names" on page 117. Next come one or more hunks of differences; each hunk shows one area where the files differ. Unified format hunks look like the following

```
@@ from-file-range to-file-range @@
line-from-either-file
line-from-either-file...
```

The lines common to both files begin with a space character. The lines that actually differ between the two files have one of the following indicator characters in the left column:

'+' A line was added here to the first file.

'-' A line was removed here from the first file.

An example of unified format

Here is the output of the command, '`diff -u lao tzu`' (see "Two sample input files" on page 110 for the complete contents of the two files):

```
--- lao Sat Jan 26 23:30:39 1991
+++ tzu Sat Jan 26 23:30:50 1991
@@ -1,7 +1,6 @@
-The Way that can be told of is not the eternal Way;
-The name that can be named is not the eternal name.
 The Nameless is the origin of Heaven and Earth;
-The Named is the mother of all things.
+The named is the mother of all things.
+
 Therefore let there always be non-being,
   so we may see their subtlety,
 And let there always be being,
@@ -9,3 +8,6 @@
 The two are the same,
 But after they are produced,
   they have different names.
+They both may be called deep and profound.
+Deeper and more profound,
+The door of all subtleties!
```

Showing which sections differences are in

Sometimes you might want to know which part of the files each change falls in. If the files are source code, this could mean which function was changed. If the files are documents, it could mean which chapter or appendix was changed. GNU `diff` can show this by displaying the nearest section heading line that precedes the differing lines. Which lines are "section headings" is determined by a regular expression.

Showing lines that match regular expressions

To show in which sections differences occur for files that are not source code for C or similar languages, use the '`-F` *regexp*' or '`--show-function-line=`*regexp*' option. `diff` considers lines that match the argument, *regexp*, to be the beginning of a section of the file. Here are suggested regular expressions for some common languages:

C, C++, Prolog
'`^[A-Za-z_]`'

Lisp
'`^(`'

Texinfo
'`^@\(chapter\|appendix\|unnumbered\|chapheading\)`'

This option does not automatically select an output format; in order to use it, you must select the context format (see "Context format" on page 113) or unified format (see "Unified format" on page 115). In other output formats it has no effect.

The '`-F`' and '`--show-function-line`' options find the nearest unchanged line that precedes each hunk of differences and matches the given regular expression. Then they add that line to the end of the line of asterisks in the context format, or to the '`@@`' line in unified format. If no matching line exists, they leave the output for that hunk unchanged. If that line is more than 40 characters long, they output only the first 40 characters. You can specify more than one regular expression for such lines; `diff` tries to match each line against each regular expression, starting with the last one given. This means that you can use '`-p`' and '`-F`' together, if you wish.

Showing C function headings

To show in which functions differences occur for C and similar languages, you can use the '`-p`' or '`--show-c-function`' option. This option automatically defaults to the context output format (see "Context format" on page 113), with the default number of lines of context. You can override that number with '`-C` *lines*' elsewhere in the command line. You can override both the format and the number with '`-U` *lines*' elsewhere in the command line.

The '`-p`' and '`--show-c-function`' options are equivalent to '`-F'^[_a-zA-Z$]'`' if the unified format is specified, otherwise '`-c -F'^[_a-zA-Z$]'`' (see "Showing lines that match regular expressions" on page 117).

GNU `diff` provides them for the sake of convenience.

Showing alternate file names

If you are comparing two files that have meaningless or uninformative names, you

might want `diff` to show alternate names in the header of the context and unified output formats.

To do this, use the '`-L` *`label`*' or '`--label=`*`label`*' option. The first time you give this option, its argument replaces the name and date of the first file in the header; the second time, its argument replaces the name and date of the second file. If you give this option more than twice, `diff` reports an error. The '`-L`' option does not affect the file names in the `pr` header when the '`-l`' or'`--paginate`' option is used (see "Paginating diff output" on page 135). The following are the first two lines of the output from '`diff -C2 -Loriginal -Lmodified lao tzu`':

```
*** original
--- modified
```

Showing differences side by side

`diff` can produce a side by side difference listing of two files. The files are listed in two columns with a gutter between them. The gutter contains one of the following markers:

white space
: The corresponding lines are in common. That is, either the lines are identical, or the difference is ignored because of one of the '`--ignore`' options (see "Suppressing differences in blank and tab spacing" on page 102).

'`|`'
: The corresponding lines differ, and they are either both com-plete or both incomplete.

'`<`'
: The files differ and only the first file contains the line.

'`>`'
: The files differ and only the second file contains the line.

'`(`'
: Only the first file contains the line, but the difference is ig-nored.

'`)`'
: Only the second file contains the line, but the difference is ignored.

'`\`'
: The corresponding lines differ, and only the first line is in-complete.

'`/`'
: The corresponding lines differ, and only the second line is incomplete.

Normally, an output line is incomplete if and only if the lines that it contains are incomplete; see "Incomplete lines" on page 197. However, when an output line represents two differing lines, one might be incomplete while the other is not. In this case, the output line is complete, but its the gutter is marked '`\`' if the first line is incomplete, '`/`' if the second line is.

Side by side format is sometimes easiest to read, but it has limitations. It generates much wider output than usual, and truncates lines that are too long to fit. Also, it relies on lining up output more heavily than usual, so its output looks particularly bad if you use varying width fonts, nonstandard tab stops, or nonprinting characters.

You can use the `sdiff` command to interactively merge side by side differences. See "Interactive merging with sdiff" on page 153 for more information on merging files.

Controlling side by side format

The ‘`-y`’ or ‘`--side-by-side`’ option selects side by side format. Because side by side output lines contain two input lines, they are wider than usual. They are normally 130 columns, which can fit onto a traditional printer line. You can set the length of output lines with the ‘`-W columns`’ or ‘`--width=`*`columns`*’ option. The output line is split into two halves of equal length, separated by a small gutter to mark differences; the right half is aligned to a tab stop so that tabs line up. Input lines that are too long to fit in half of an output line are truncated for output. The ‘`--left-column`’ option prints only the left column of two common lines. The ‘`--suppress-common-lines`’ option suppresses common lines entirely.

An example of side by side format

The following is the output of the command ‘`diff -y -W 72 lao tzu`’ (see “Two sample input files” on page 110 for the complete contents of the two files).

```
The Way that can be told of is <
The name that can be named is <
The Nameless is the origin of   The Nameless is the origin of
The Named is the mother of all  | The named is the mother of all
                                >
Therefore let there always be     Therefore let there always be
  so we may see their subtlet     so we may see their subtlet
And let there always be being   And let there always be being
  so we may see their outcome     so we may see their outcome
The two are the same,             The two are the same,
But after they are produced,      But after they are produced,
they have different names.          they have different names.
                                >  They both may be called deep
                                >  Deeper and more profound,
                                 >  The door of all subtleties!
```

Making edit scripts

Several output modes produce command scripts for editing *`from-file`* to produce *`to-file`*.

ed scripts

`diff` can produce commands that direct the `ed` text editor to change the first file into the second file. Long ago, this was the only output mode that was suitable for editing one file into another automatically; today, with `patch`, it is almost obsolete. Use the ‘`-e`’ or‘`--ed`’ option to select this output format. Like the normal format (see “Showing differences without context” on page 111), this output format does not show

any context; unlike the normal format, it does not include the information necessary to apply the diff in reverse (to produce the first file if all you have is the second file and the diff). If the file ‘d’ contains the output of ‘diff -e old new’, then the command, ‘(cat d && echo w) | ed - old’, edits ‘old’ to make it a copy of ‘new’.

More generally, if ‘d1’, ‘d2’, . . .,‘dN’ contain the outputs of ‘diff -e old new1’, ‘diff -e new1 new2’, . . .,‘diff -e newN-1 newN’, respectively, then the command, ‘(cat d1 d2 ...dN && echo w) | ed - old’, edits ‘old’ to make it a copy of ‘newN’.

Detailed description of ed format

The ed output format consists of one or more hunks of differences. The changes closest to the ends of the files come first so that commands that change the number of lines do not affect how ed interprets line numbers in succeeding commands. ed format hunks look like the following:

```
change-command
to-file-line
to-file-line...
```

Because ed uses a single period on a line to indicate the end of input,

GNU diff protects lines of changes that contain a single period on a line by writing two periods instead, then writing a subsequent ed command to change the two periods into one. The ed format cannot represent an incomplete line, so if the second file ends in a changed incomplete line, diff reports an error and then pretends that a newline was appended.

There are three types of change commands. Each consists of a line number or comma-separated range of lines in the first file and a single character indicating the kind of change to make. All line numbers are the original line numbers in the file. The types of change commands are:

‘*l*a’
: Add text from the second file after line *l* in the first file. For example, ‘8a’ means to add the following lines after line 8 of file 1.

‘*r*c’
: Replace the lines in range *r* in the first file with the following lines. Like a combined add and delete, but more compact. For example, ‘5,7c’ means change lines 5–7 of file 1 to read as the text file 2.

‘*r*d’
: Delete the lines in range r from the first file. For example, ‘5,7d’ means delete lines 5–7 of file 1.

Example ed Script

The following is the output of ‘`diff -e lao tzu`’ (see “Two sample input files” on page 110 for the complete contents of the two files):

```
11a
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
.
4c
The named is the mother of all things.

.
1,2d
```

Forward ed scripts

`diff` can produce output that is like an `ed` script, but with hunks in forward (front to back) order. The format of the commands is also changed slightly: command characters precede the lines they modify, spaces separate line numbers in ranges, and no attempt is made to disambiguate hunk lines consisting of a single period. Like `ed` format, forward `ed` format cannot represent incomplete lines. Forward `ed` format is not very useful, because neither `ed` nor `patch` can apply diffs in this format. It exists mainly for compatibility with older versions of diff. Use the ‘`-f`’ or ‘`--forward-ed`’ option to select it.

RCS scripts

The RCS output format is designed specifically for use by the Revision Control System, which is a set of free programs used for organizing different versions and systems of files. Use the ‘`-n`’ or ‘`--rcs`’ option to select this output format. It is like the forward `ed` format (see “Forward ed scripts” on page 122), but it can represent arbitrary changes to the contents of a file because it avoids the forward `ed` format’s problems with lines consisting of a single period and with incomplete lines. Instead of ending text sections with a line consisting of a single period, each command specifies the number of lines it affects; a combination of the ‘`a`’ and ‘`d`’ commands are used instead of ‘`c`’. Also, if the second file ends in a changed incomplete line, then the output also ends in an incomplete line. The following is the output of ‘`diff -n lao tzu`’ (see “Two sample input files” on page 110 for the complete contents of the two files):

```
d1  2
d4  1
a4  2
The named is the mother of all things.
```

```
a11 3
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
```

Merging files with if-then-else

You can use `diff` to merge two files of C source code. The output of `diff` in this format contains all the lines of both files. Lines common to both files are output just once; the differing parts are separated by the C preprocessor directives, `#ifdef` *name* or `#ifndef` *name*, `#else`, and `#endif`. When compiling the output, you select which version to use by either defining or leaving undefined the macro name.

To merge two files, use `diff` with the '`-D` *name*' or '`--ifdef=`*name*' option. The argument, *name*, is the C preprocessor identifier to use in the `#ifdef` and `#ifndef` directives. For example, if you change an instance of `wait (&s)` to `waitpid (-1, &s, 0)` and then merge the old and new files with the '`--ifdef=HAVE_WAITPID`' option, then the affected part of your code might look like the following declaration.

```
   do {
#ifndef HAVE_WAITPID
      if ((w = wait (&s)) < 0 && errno != EINTR)
#else /* HAVE_WAITPID */
if ((w = waitpid (-1, &s, 0)) < 0 && errno != EINTR)
#endif /* HAVE_WAITPID */
         return w; }
   while (w != child);
```

You can specify formats for languages other than C by using line group formats and line formats.

Line group formats

Line group formats let you specify formats suitable for many applications that allow if-then-else input, including programming languages and text formatting languages. A line group format specifies the output format for a contiguous group of similar lines. For example, the following command compares the TeX files '`old`' and '`new`', and outputs a merged file in which old regions are surrounded by '`\begin{em}`'-'`\end{em}`' lines, and new regions are surrounded by '`\begin{bf}`'-'`\end{bf}`' lines.

```
diff \
  --old-group-format='\begin{em}
%<\end{em}
' \
  --new-group-format='\begin{bf}
%>\end{bf}

' \
  old new
```

The following command is equivalent to the previous example, but it is a little more

verbose, because it spells out the default line group formats.

```
diff \
   --old-group-format='\begin{em}
%<\end{em}
' \
   --new-group-format='\begin{bf}
%>\end{bf} '
\
   --unchanged-group-format='%=' \
   --changed-group-format='\begin{em}
%<\end{em}
\begin{bf}
%>\end{bf}
' \
   old new
```

Here is a more advanced example, which outputs a diff listing with headers containing line numbers in a “plain English” style.

```
diff \
--unchanged-group-format='' \
 --old-group-format='-------- %dn line%(n=1?:s) deleted at %df:
%<' \
 --new-group-format='-------- %dN line%(N=1?:s) added after %de:
%>' \
 --changed-group-format='-------- %dn line%(n=1?:s) changed at %df:
%<-------- to:
%>' \
 old new
```

To specify a line group format, use `diff` with one of the options listed below. You can specify up to four line group formats, one for each kind of line group. You should quote *`format`*, because it typically contains shell metacharacters.

‘`--old-group-format=`*`format`*’
: These line groups are hunks containing only lines from the first file. The default old group format is the same as the changed group format if it is specified; otherwise it is a format that outputs the line group as-is.

‘`--new-group-format=`*`format`*’
: These line groups are hunks containing only lines from the second file. The default new group format is same as the the changed group format if it is specified; otherwise it is a format that outputs the line group as-is.

‘`--changed-group-format=`*`format`*’
: These line groups are hunks containing lines from both files. The default changed group format is the concatenation of the old and new group formats.

‘`--unchanged-group-format=`*`format`*’
These line groups contain lines common to both files. The default unchanged group format is a format that outputs the line group as-is.

In a line group format, ordinary characters represent themselves; conversion specifications start with ‘%’ and have one of the following forms.

‘`%<`’
Stands for the lines from the first file, including the trailing newline. Each line is formatted according to the old line format (see “Line formats” on page 127).

‘`%>`’
Stands for the lines from the second file, including the trailing newline. Each line is formatted according to the new line format.

‘`%=`’
Stands for the lines common to both files, including the trail-ing newline. Each line is formatted according to the unchanged line format.

‘`%%`’
Stands for ‘%’.

‘`%c' `*`C`*`'`’
Where *`C`* is a single character, stands for *`C`*. *`C`* may not be a backslash or an apostrophe. For example, ‘`%c':'`’ stands for a colon, even inside the ‘then-’ part of an if-then-else format, which a colon would normally terminate.

‘`%c'\ `*`O`*`'`’
Stands for the character with octal code *`O`* ,where *`O`* is a string of 1, 2, or 3 octal digits. For example, ‘`%c'\0'`’ stands for a null character.

‘*`Fn`*’
Stands for n’s value formatted with *`F`* where *`F`* is a `printf` conversion specification and *`n`* is one of the following letters.

‘*`e`*’
The line number of the line just before the group in the old file.

‘*`f`*’
The line number of the first line in the group in the old file; equals *`e`* + 1.

‘*`l`*’
The line number of the last line in the group in the old file.

‘*`m`*’
The line number of the line just after the group in the old file; equals *`l`* + 1.

‘*`n`*’
The number of lines in the group in the old file; equals *`l`*-*`f`* + 1.

‘*E, F, L, M, N*’
Likewise, for lines in the new file.

The `printf` conversion specification can be ‘`%d`’, ‘`%o`’, ‘`%x`’, or ‘`%X`’, specifying decimal, octal, lower case hexadecimal, or upper case hexadecimal output respectively. After the ‘`%`’ the following options can appear in sequence: a ‘`-`’ specifying left-justification; an integer specifying the minimum field width; and a period followed by an optional integer specifying the minimum number of digits. For example, ‘`%5dN`’ prints the number of new lines in the group in a field of width 5 characters, using the `printf` format, `"%5d"`.

‘`(`*A*`=`*B*`?`*T*`:`*E*`)`’
If *A* equals *B*, then *T*, else, *E*. *A* and *B* are each either a decimal constant or a single letter interpreted as above. This format spec is equivalent to *T* if *A*’s value equals *B*’s; otherwise it is equivalent to *E*. For example, ‘`%(N=0?no:%dN) line%(N=1?:s)`’ is equivalent to ‘`no lines`’ if *N* (the number of lines in the group in the the new file) is 0, to ‘`1 line`’ if *N* is 1, and to ‘`%dN lines`’ otherwise.

Line formats

Line formats control how each line taken from an input file is output as part of a line group in if-then-else format. For example, the following command outputs text with a one-column change indicator to the left of the text. The first column of output is ‘`-`’ for deleted lines, ‘`|`’ for added lines, and a space for unchanged lines. The formats contain newline characters where newlines are desired on output.

```
diff \
   --old-line-format='-%l
' \
   --new-line-format='|%l
' \
   --unchanged-line-format=' %l
' \
   old new
```

To specify a line format, use one of the following options. You should quote *format*, since it often contains shell metacharacters.

‘**`--old-line-format=`***format*’
Formats lines just from the first file.

‘**`--new-line-format=`***format*’
Formats lines just from the second file.

‘**`--unchanged-line-format=`***format*’
Formats lines common to both files.

‘**`--line-format=`***format*’
Formats all lines; in effect, it simultaneously sets all three of the previous options.

In a line format, ordinary characters represent themselves; conversion specifications start with ‘`%`’ and have one of the following forms.

‘`%l`’
Stands for the the contents of the line, not counting its trail-ing newline (if any). This format ignores whether the line is incomplete; see “Incomplete lines” on page 197.

‘`%L`’
Stands for the the contents of the line, including its trailing newline (if any). If a line is incomplete, this format preserves its incompleteness.

‘`%%`’
Stands for ‘`%`’.

‘`%c' C'`’
Stands for *`C`*, where *`C`* is a single character. *`C`* may not be a backslash or an apostrophe. For example, ‘`%c':'`’ stands for a colon.

‘`%c'\ O'`’
Stands for the character with octal code *`O`* where *`O`* is a string of 1, 2, or 3 octal digits. For example, ‘`%c'\0'`’ stands for a null character.

‘*`Fn`*’
Stands for the line number formatted with *`F`* where *`F`* is a `printf` conversion specification. For example, ‘`%.5dn`’ prints the line number using the `printf` format, `“%.5d”`. See “Line group formats” on page 124 for more about `printf` conversion specifications.

The default line format is ‘`%l`’ followed by a newline character.

If the input contains tab characters and it is important that they line up on output, you should ensure that ‘`%l`’ or ‘`%L`’ in a line format is just after a tab stop (e.g., by preceding ‘`%l`’ or ‘`%L`’ with a tab character), or you should use the ‘`-t`’ or ‘`--expand-tabs`’ option.

Taken together, the line and line group formats let you specify many different formats. For example, the following command uses a format similar to `diff`’s normal format. You can tailor this command to get fine control over `diff`’s output.

```
diff \
   --old-line-format='< %l
' \
   --new-line-format='> %l

' \
   --old-group-format='%df%(f=l?:,%dl)d%dE
%<' \
   --new-group-format='%dea%dF%(F=L?:,%dL)
%>' \
```

```
   --changed-group-format='%df%(f=l?:,%dl)c%dF%(F=L?:,%dL)
%<---
%>' \
   --unchanged-group-format='' \
   old new
```

Detailed description of if-then-else format

For lines common to both files, `diff` uses the unchanged line group format. For each hunk of differences in the merged output format, if the hunk contains only lines from the first file, `diff` uses the old line group format; if the hunk contains only lines from the second file, diff uses the new group format; otherwise, `diff` uses the changed group format.

The old, new, and unchanged line formats specify the output format of lines from the first file, lines from the second file, and lines common to both files, respectively.

The option '`--ifdef=`*name*' is equivalent to the following sequence of options using shell syntax:

```
--old-group-format='#ifndef name %<#endif /* not name */ ' \
--new-group-format='#ifdef name %>#endif /* name */ ' \
--unchanged-group-format='%=' \ --changed-group-format='#ifndef name
%<#else /* name */ %>#endif /* name */ '
```

You should carefully check the `diff` output for proper nesting. For example, when using the the '`-D` *name*' or '`--ifdef=`*name*' option, you should check that if the differing lines contain any of the C preproces-sor directives '`#ifdef`', '`#ifndef`', '`#else`', '`#elif`', or '`#endif`', they are nested properly and match. If they don't, you must make corrections manually. It is a good idea to carefully check the resulting code any-way to make sure that it really does what you want it to; depending on how the input files were produced, the output might contain duplicate or otherwise incorrect code.

The `patch` '`-D` *name*' option behaves just like the `diff` '`-D` *name*' option, except it operates on a file and a diff to produce a merged file; see "Options to patch" on page 189.

An example of if-then-else format

The following is the output of '`diff -DTWO lao tzu`' (see "Two sample input files" on page 110 for the complete contents of the two files):

```
#ifndef TWO
The Way that can be told of is not the eternal Way;
The name that can be named is not the eternal name.
#endif /* not TWO */
The Nameless is the origin of Heaven and Earth;
```

```
#ifndef TWO
The Named is the mother of all things.
#else /* TWO */
The named is the mother of all things.

#endif /* TWO */
Therefore let there always be non-being,
so we may see their subtlety,
And let there always be being,
so we may see their outcome.
The two are the same,
But after they are produced,
they have different names.
#ifdef TWO
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
#endif /* TWO */
```

4

Comparing directories

You can use `diff` to compare some or all of the files in two directory trees. When both file name arguments to `diff` are directories, it compares each file that is contained in both directories, ex-amining file names in alphabetical order. Normally `diff` is silent about pairs of files that contain no differences, but if you use the '`-s`' or'`--report-identical-files`' option, it reports pairs of identical files. Normally diff reports subdirectories common to both directories without comparing subdirectories' files, but if you use the '`-r`' or '`--recursive`' option, it compares every corresponding pair of files in the directory trees, as many levels deep as they go.

For file names that are in only one of the directories, `diff` normally does not show the contents of the file that exists; it reports only that the file exists in that directory and not in the other. You can make `diff` act as though the file existed but was empty in the other directory, so that it outputs the entire contents of the file that actually exists. (It is output as either an insertion or a deletion, depending on whether it is in the first or the second directory given.) To do this, use the '`-N`' or '`--new-file`' option.

If the older directory contains one or more large files that are not in the newer directory, you can make the patch smaller by using the '`-P`' or '`--unidirectional-new-file`' options instead of '`-N`'. This option is like '`-N`' except that it only inserts the contents of files that appear in the second directory but not the first (that is, files that were added). At the top of the patch, write instructions

for the user applying the patch to remove the files that were deleted before applying the patch. See “Tips for making patch distributions” on page 167 for more discussion of making patches for distribution.

To ignore some files while comparing directories, use the ‘`-x` *`pattern`*’ or ‘`--exclude=`*`pattern`*’ option. This option ignores any files or subdi-rectories whose base names match the shell pattern pattern. Unlike in the shell, a period at the start of the base of a file name matches a wildcard at the start of a pattern. You should enclose pattern in quotes so that the shell does not expand it. For example, the option ‘`-x '*.[ao]'`’ ignores any file whose name ends with ‘`.a`’ or ‘`.o`’.

This option accumulates if you specify it more than once. For example, using the options ‘`-x 'RCS' -x '*,v'`’ ignores any file or subdirectory whose base name is ‘`RCS`’ or ends with ‘`,v`’.

If you need to give this option many times, you can instead put the patterns in a file, one pattern per line, using the ‘`-X` *`file'`* or ‘`--exclude-from-`*`file`*’ option.

If you have been comparing two directories and stopped partway through, later you might want to continue where you left off. You can do this by using the ‘`-S` *`file`*’ or‘`--starting-file-`*`file`*’ option. This compares only the file, *`file`*, and all alphabetically subsequent files in the topmost directory level.

5

Making `diff` output prettier

diff provides several ways to adjust the appearance of its output. These adjustments can be applied to any output format. For more information, see “Preserving tabstop alignment” on page 134 and “Paginating diff output” on page 135.

Preserving tabstop alignment

The lines of text in some of the diff output formats are preceded by one or two characters that indicate whether the text is inserted, deleted, or changed. The addition of those characters can cause tabs to move to the next tabstop, throwing off the alignment of columns in the line. GNU `diff` provides two ways to make tab-aligned columns line up correctly.

The first way is to have `diff` convert all tabs into the correct number of spaces before outputting them.

Select this method with the ‘`-t`’ or ‘`--expand-tabs`’ option. `diff` assumes that tabstops are set every 8 columns. To use this form of output with `patch`, use the ‘`-l`’ or ‘`--ignore-white-space`’ option (see “Applying Patches in Other Directories” on page 185 for more information).

The other method for making tabs line up correctly is to add a tab character instead of a space after the indicator character at the beginning of the line. This ensures that all following tab characters are in the same position relative to tabstops that they were in the original files, so that the output is aligned correctly. Its disadvantage is that it can make long lines too long to fit on one line of the screen or the paper. It also does not work with the unified output format which does not have a space character after the change type indicator character.

Select this method with the ‘`-T`’ or ‘`--initial-tab`’ option.

Paginating `diff` output

It can be convenient to have long output page-numbered and time-stamped. The '`-l`' and '`--paginate`' options do this by sending the `diff` output through the `pr` program. The following is what the page header might look like for '`diff -lc lao tzu`':

```
Mar 11 13:37 1991 diff -lc lao tzu Page 1
```

6

`diff` performance tradeoffs

GNU `diff` runs quite efficiently; however, in some circumstances you can cause it to run faster or produce a more compact set of changes. There are two ways that you can affect the performance of GNU `diff` by changing the way it compares files.

Performance has more than one dimension. These options improve one aspect of performance at the cost of another, or they improve performance in some cases while hurting it in others.

The way that GNU `diff` determines which lines have changed always comes up with a near-minimal set of differences. Usually it is good enough for practical purposes. If the `diff` output is large, you might want `diff` to use a modified algorithm that sometimes produces a smaller set of differences. The '`-d`' or '`--minimal`' option does this; however, it can also cause `diff` to run more slowly than usual, so it is not the default behavior.

When the files you are comparing are large and have small groups of changes scattered throughout them, use the '`-H`' or '`--speed-large-files`' option to make a different modification to the algorithm that `diff` uses. If the input files have a constant small density of changes, this option speeds up the comparisons without changing the output. If not, `diff` might produce a larger set of differences; however, the output will still be correct.

Normally `diff` discards the prefix and suffix that is common to both files before it attempts to find a minimal set of differences. This makes `diff` run faster, but occasionally it may produce non-minimal output.

The '`--horizon-lines=` *`lines`*' option prevents `diff` from discarding the last *`lines`* lines of the prefix and the first *`lines`* lines of the suffix. This gives `diff` further opportunities to find a minimal output.

7

Comparing three files

Use the program `diff3` to compare three files and show any differences among them. (`diff3` can also merge files; see “Merging from a common ancestor” on page 145).

The “normal” `diff3` output format shows each hunk of differences without surrounding context. Hunks are labeled depending on whether they are two-way or three-way, and lines are annotated by their location in the input files.

See “Invoking diff3” on page 179 for more information on how to run `diff3`.

The following documentation discusses comparing files.

A third sample input file

The following is a third sample file that will be used in examples to illustrate the output of `diff3` and how various options can change it. The first two files are the same that we used for `diff` (see “Two sample input files” on page 110). This is the third sample file, called ‘`tao`’:

```
The Way that can be told of is not the eternal Way;
The name that can be named is not the eternal name.
The Nameless is the origin of Heaven and Earth;
The named is the mother of all things.

Therefore let there always be non-being,
  so we may see their subtlety,
And let there always be being,
  so we may see their result.
The two are the same,
But after they are produced,
  they have different names.

   -- The Way of Lao-Tzu, tr. Wing-tsit Chan
```

Detailed description of diff3 normal format

Each hunk begins with a line marked '====='. Three-way hunks have plain '====' lines, and two-way hunks have '1', '2', or '3' appended to specify which of the three input files differ in that hunk. The hunks contain copies of two or three sets of input lines each preceded by one or two commands identifying the origin of the lines.

Normally, two spaces precede each copy of an input line to distinguish it from the commands. But with the '-T' or'--initial-tab' option, diff3 uses a tab instead of two spaces; this lines up tabs correctly.

See "Preserving tabstop alignment" on page 134 for more information.

Commands take the following forms.

'*file*: *l*a'
: This hunk appears after line *l* of file, *file*, and contains no lines in that file. To edit this file to yield the other files, one must append hunk lines taken from the other files. For example, '1:11a' means that the hunk follows line 11 in the first file and contains no lines from that file.

'*file*: *r*c'
: This hunk contains the lines in the range r of file file. The range r is a comma-separated pair of line numbers, or just one number if the range is a singleton. To edit this file to yield the other files, one must change the specified lines to be the lines taken from the other files. For example, '2:11,13c' means that the hunk contains lines 11 through 13 from the second file.

If the last line in a set of input lines is incomplete, it is distinguished on output from a full line by a following line that starts with '\' (see "Incomplete lines" on page 197).

diff3 hunks

Groups of lines that differ in two or three of the input files are called diff3 hunks, by analogy with diff hunks (see “Hunks” on page 101). If all three input files differ in a diff3 hunk, the hunk is called a three-way hunk; if just two input files differ, it is a two-way hunk. As with diff, several solutions are possible. When comparing the files ‘A’, ‘B’, and ‘C’, diff3 normally finds diff3 hunks by merging the two-way hunks output by the two commands ‘diff A B’ and ‘diff A C’. This does not necessarily minimize the size of the output, but exceptions should be rare. For example, suppose ‘F’ contains the three lines ‘a’, ‘b’, ‘f’, ‘G’ contains the lines ‘g’, ‘b’, ‘g’, and ‘H’ contains the lines ‘a’, ‘b’, ‘h’. ‘diff3 F G H’ might output the following:

```
====2
1:1c
3:1c
a
2:1c
g
====
1:3c
f
2:3c
g
3:3c
h
```

because it found a two-way hunk containing ‘a’ in the first and third files and ‘g’ in the second file, then the single line ‘b’ common to all three files, then a three-way hunk containing the last line of each file.

An example of `diff3` normal format

Here is the output of the command ‘`diff3 lao tzu tao`’ (see “A third sample input file” on page 140 for the complete contents of the files). Notice that it shows only the lines that are different among the three files.

```
====2
1:1,2c
3:1,2c
  The Way that can be told of is not the eternal Way;
  The name that can be named is not the eternal name.
2:0a
====
1 1:4c
  The Named is the mother of all things.
2:2,3c
3:4,5c
  The named is the mother of all things.

====3
1:8c
2:7c
    so we may see their outcome.
3:9c
    so we may see their result.
====
1:11a
2:11,13c
  They both may be called deep and profound.
  Deeper and more profound,
  The door of all subtleties!
3:13,14c

    -- The Way of Lao-Tzu, tr. Wing-tsit Chan
```

8

Merging from a common ancestor

When two people have made changes to copies of the same file, `diff3` can produce a merged output that contains both sets of changes together with warnings about conflicts. One might imagine programs with names like `diff4` and `diff5` to compare more than three files simultaneously, but in practice the need rarely arises. You can use `diff3` to merge three or more sets of changes to a file by merging two change sets at a time.

`diff3` can incorporate changes from two modified versions into a common preceding version. This lets you merge the sets of changes represented by the two newer files. Specify the common ancestor version as the second argument and the two newer versions as the first and third arguments, like this: `diff3` *`mine older yours`*.

You can remember the order of the arguments by noting that they are in alphabetical order.

You can think of this as subtracting *`older`* from yours and adding the result to *`mine`*, or as merging into *`mine`* the changes that would turn *`older`* into *`yours`*. This merging is well-defined as long as *`mine`* and *`older`* match in the neighborhood of each such change. This fails to be true when all three input files differ or when only *`older`* differs; we call this a *conflict*. When all three input files differ, we call the conflict an *overlap*.

`diff3` gives you several ways to handle overlaps and conflicts. You can omit overlaps or conflicts, or select only overlaps, or mark conflicts with special '`<<<<<<<`' and '`>>>>>>>`' lines.

`diff3` can output the merge results as an `ed` script that that can be applied to the first file to yield the merged output. However, it is usually better to have `diff3` generate the merged output directly; this bypasses some problems with `ed`.

Selecting which changes to incorporate

You can select all unmerged changes from olderto yoursfor merging into minewith the ‘`-e`’ or ‘`--ed`’ option. You can select only the nonover-lapping unmerged changes with ‘`-3`’ or ‘`--easy-only`’, and you can select only the overlapping changes with ‘`-x`’ or ‘`--overlap-only`’.

The ‘`-e`’, ‘`-3`’ and ‘`-x`’ options select only unmerged changes, i.e., changes where *`mine`* and *`yours`* differ; they ignore changes from *`older`* to *`yours`* where *`mine`* and *`yoursare`* identical, because they assume that such changes have already been merged. If this assumption is not a safe one, you can use the options, ‘`-A`’ or ‘`--show-all`’ (see “Marking conflicts” on page 148). The following is the output of the command, `diff3`, with each of these three options (see “A third sample input file” on page 140 for the complete contents of the files). Notice that ‘`-e`’ outputs the union of the disjoint sets of changes output by ‘`-3`’ and ‘`-x`’.

Output of ‘`diff3 -e lao tzu tao`’:

```
11a

  -- The Way of Lao-Tzu, tr. Wing-tsit Chan
.
8c
  so we may see their result.
.
```

Output of ‘`diff3 -3 lao tzu tao`’:

```
8c
  so we may see their result.
.
```

Output of ‘`diff3 -x lao tzu tao`’:

```
11a

-- The Way of Lao-Tzu, tr. Wing-tsit Chan
.
```

Marking conflicts

`diff3` can mark conflicts in the merged output by bracketing them with special marker lines. A conflict that comes from two files *`A`* and *`B`* is marked as follows:

```
<<<<<<< A
lines from A
=======
lines from B
>>>>>>> B
```

A conflict that comes from three files, *`A`*, *`B`* and *`C`*, is marked as follows:

```
<<<<<<< A
lines from A
||||||| B
lines from B
=======
lines from C
>>>>>>> C
```

The ‘`-A`’ or ‘`--show-all`’ option acts like the ‘`-e`’ option, except that it brackets conflicts, and it outputs all changes from *`older`* to *`yours`*, not just the unmerged changes. Thus, given the sample input files (see “A third sample input file” on page 140), ‘`diff3 -A lao tzu tao`’ puts brackets around the conflict where only ‘`tzu`’ differs:

```
<<<<<<< tzu
=======
The Way that can be told of is not the eternal Way;
The name that can be named is not the eternal name.
>>>>>>> tao
```

And it outputs the three-way conflict as follows:

```
<<<<<<< lao
||||||| tzu
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
=======

-- The Way of Lao-Tzu, tr. Wing-tsit Chan
>>>>>>> tao
```

The ‘`-E`’ or‘`--show-overlap`’ option outputs less information than the ‘`-A`’ or ‘`--show-all`’ option, because it outputs only unmerged changes, and it never outputs the contents of the second file. Thus the ‘`-E`’ option acts like the ‘`-e`’ option, except that it brackets the first and third files from three-way overlapping changes. Similarly, ‘`-X`’ acts like ‘`-x`’, except it brackets all its (necessarily overlapping) changes. For

example, for the three-way overlapping change above, the '`-E`' and '`-X`' options output the following:

```
<<<<<<< lao
 =======

  -- The Way of Lao-Tzu, tr. Wing-tsit Chan
>>>>>>> tao
```

If you are comparing files that have meaningless or uninformative names, you can use the '`-L` *`label`*' or '`--label=` *`label`*' option to show alternate names in the '`<<<<<<<`', '`|||||||`' and '`>>>>>>>`' brackets. This option can be given up to three times, once for each input file. Thus '`diff3 -A -L X -L Y -L Z A B C`' acts like '`diff3 -A A B C`', except that the output looks like it came from files named '`X`', '`Y`' and '`Z`' rather than from files named '`A`', '`B`' and '`C`'.

Generating the merged output directly

With the ‘`-m`’ or ‘`--merge`’ option, `diff3` outputs the merged file directly. This is more efficient than using ed to generate it, and works even with non-text files that `ed` would reject. If you specify ‘`-m`’ without an `ed` script option, ‘`-A`’ (‘`--show-all`’) is assumed.

For example, the command ‘`diff3 -m lao tzu tao`’ (see “A third sample input file” on page 140 for a copy of the input files) would output the following:

```
<<<<<<< tzu
=======
The Way that can be told of is not the eternal Way;
The name that can be named is not the eternal name.
>>>>>>> tao
The Nameless is the origin of Heaven and Earth;
The Named is the mother of all things.
Therefore let there always be non-being,
  so we may see their subtlety,
And let there always be being,
  so we may see their result.
The two are the same,
But after they are produced,
  they have different names.
<<<<<<< lao
||||||| tzu
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
=======

-- The Way of Lao-Tzu, tr. Wing-tsit Chan
>>>>>>> tao
```

How diff3 merges incomplete lines

With '-m', incomplete lines (see"Incomplete lines" on page 197) are simply copied to the output as they are found; if the merged output ends in an conflict and one of the input files ends in an incomplete line, succeeding '|||||||', '=======' or'>>>>>>>' brackets appear somewhere other than the start of a line because they are appended to the incomplete line.

Without '-m', if an ed script option is specified and an incomplete line is found, diff3 generates a warning and acts as if a newline had been present.

Saving the changed file

Traditional Unix `diff3` generates an `ed` script without the trailing ‘`w`’ and and ‘`q`’ commands that save the changes. System V `diff3` generates these extra commands. GNU `diff3` normally behaves like traditional Unix `diff3`, but with the ‘`-i`’ option it behaves like System V `diff3` and appends the ‘`w`’ and ‘`q`’ commands.

The ‘`-i`’ option requires one of the `ed` script options ‘`-AeExX3`’, and is incompatible with the merged output option ‘`-m`’.

9

Interactive merging with `sdiff`

With `sdiff`, you can merge two files interactively based on a side-by-side format comparison with ‘`-y`’ (see “Showing differences side by side” on page 119). Use ‘`-o` *`file`*’ or ‘`--output=`*`file`*’ to specify where to put the merged text. See “Invoking sdiff” on page 193 for more details on the options to `sdiff`.

Another way to merge files interactively is to use the Emacs Lisp package, `emerge`. See “Merging Files with Emerge” in ***The GNU Emacs Manual*** for more information.

Specifying diff options to sdiff

The following sdiff options have the same meaning as for diff. See “Options to diff” on page 172 for the use of these options.

```
-a -b -d -i -t -v
-B -H -I regexp

--ignore-blank-lines --ignore-case
--ignore-matching-lines= regexp --ignore-space-change
--left-column --minimal --speed-large-files
--suppress-common-lines --expand-tabs
--text --version --width= columns
```

For historical reasons, sdiff has alternate names for some options. The ‘-l’ option is equivalent to the ‘--left-column’ option, and similarly ‘-s’ is equivalent to ‘--suppress-common-lines’. The meaning of the sdiff ‘-w’ and ‘-W’ options is interchanged from that of diff: with sdiff, ‘-w *columns*’ is equivalent to ‘--width=*columns*’, and ‘-W’ is equivalent to ‘--ignore-all-space’. sdiff without the ‘-o’ option is equivalent to diff with the ‘-y’ or ‘--side-by-side’ option (see “Showing differences side by side” on page 119).

Merge commands

Groups of common lines, with a blank gutter, are copied from the first file to the output. After each group of differing lines, `sdiff` prompts with '`%`' and pauses, waiting for one of the following commands. Follow each command using the Enter key.

'`e`'
: Discard both versions. Invoke a text editor on an empty temporary file, then copy the resulting file to the output.

'`eb`'
: Concatenate the two versions, edit the result in a temporary file, then copy the edited result to the output.

'`el`'
: Edit a copy of the left version, then copy the result to the output.

'`er`'
: Edit a copy of the right version, then copy the result to the output.

'`l`'
: Copy the left version to the output.

'`q`'
: Quit.

'`r`'
: Copy the right version to the output.

'`s`'
: Silently copy common lines.

'`v`'
: Verbosely copy common lines. This is the default.

The text editor invoked is specified by the `EDITOR` environment variable if it is set. The default is system-dependent.

10

Merging with `patch`

`patch` takes comparison output produced by `diff` and applies the differences to a copy of the original file, producing a patched version. With `patch`, you can distribute just the changes to a set of files instead of distributing the entire file set; your correspondents can apply `patch` to update their copy of the files with your changes. patch automatically determines the diff format, skips any leading or trailing headers, and uses the headers to determine which file to patch. This lets your correspondents feed an article or message containing a difference listing directly to `patch`.

`patch` detects and warns about common problems like forward patches. It saves the original version of the files it patches, and saves any patches that it could not apply. It can also maintain a `patchlevel.h` file to ensures that your correspondents apply diffs in the proper order.

`patch` accepts a series of diffs in its standard input, usually separated by headers that specify which file to patch. It applies `diff` hunks (see “Hunks” on page 101) one by one. If a hunk does not exactly match the original file, `patch` uses heuristics to try to patch the file as well as it can. If no approximate match can be found, `patch` rejects the hunk and skips to the next hunk. `patch` normally replaces each file, *`f`*, with its new version, saving the original file in ‘*`f`*`.orig`’, and putting reject hunks (if any) into ‘*`f`*`.rej`’.

See “Invoking patch” on page 183 for detailed information on the options to patch. See “Backup File Names” on page 186 for more in-formation on how patch names backup files. See “Reject File Names” on page 188 for more information on where patch puts reject hunks.

Selecting the `patch` input format

`patch` normally determines which `diff` format the patch file uses by examining its contents. For patch files that contain particularly confusing leading text, you might need to use one of the following options to force patch to interpret the `patch` file as a certain format of `diff`. The output formats shown in the following discussion are the only ones that `patch` can understand.

‘`-c`’
‘`--context`’
: context diff.

‘`-e`’
‘`--ed`’
: `ed` script.

‘`-n`’
‘`--normal`’
: normal diff.

‘`-u`’
‘`--unified`’
: unified diff.

Applying imperfect patches

`patch` tries to skip any leading text in the `patch` file, apply the diff, and then skip any trailing text. Thus you can feed a news article or mail message directly to `patch`, and it should work. If the entire diff is indented by a constant amount of white space, `patch` automatically ignores the indentation. However, certain other types of imperfect input require user intervention.

Applying patches with changed white space

Sometimes mailers, editors, or other programs change spaces into tabs, or vice versa. If this happens to a patch file or an input file, the files might look the same, but patch will not be able to match them properly. If this problem occurs, use the ‘`-l`’ or‘`--ignore-white-space`’ option, which makes patch compare white space loosely so that any sequence of white space in the patch file matches any sequence of whitespace in the input files. Non-whitespace characters must still match exactly. Each line of the context must still match a line in the input file.

Applying reversed patches

Sometimes people run `diff` with the new file first instead of second. This creates a diff that is “reversed”. To apply such patches, give patch the ‘`-R`’ or‘`--reverse`’ option. patch then attempts to swap each hunk around before applying it. Rejects come out in the swapped format. The ‘`-R`’ option does not work with `ed` scripts because there is too little information in them to reconstruct the reverse operation. Often `patch` can guess that the patch is reversed. If the first hunk of a patch fails, `patch` reverses the hunk to see if it can apply it that way. If it can, `patch` asks you if you want to have the ‘`-R`’ option set; if it can’t, `patch` continues to apply the patch normally. This method cannot detect a reversed patch if it is a normal diff and the first command is an append (which should have been a delete) since appends always succeed, because a null context matches anywhere. But most patches add or change lines rather than delete them, so most reversed normal diffs begin with a delete, which fails, and `patch` notices.

If you apply a patch that you have already applied, patch thinks it is a reversed patch and offers to un-apply the patch. This could be construed as a feature. If you did this inadvertently and you don’t want to un-apply the patch, just answer ‘`n`’ to this offer and to the subsequent “apply anyway” question—or use the keystroke sequence, **C-c**, to kill the `patch` process.

Helping patch find inexact matches

For context diffs, and to a lesser extent normal diffs, patch can detect when the line numbers mentioned in the patch are incorrect, and it attempts to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned in the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, patch scans both forward and backward for a set of lines matching the context given in the hunk.

First patch looks for a place where all lines of the context match. If it cannot find such a place, and it is reading a context or unified diff, and the maximum fuzz factor is set to 1 or more, then patch makes another scan, ignoring the first and last line of context. If that fails, and the maximum fuzz factor is set to 2 or more, it makes another scan, ignoring the first two and last two lines of context are ignored. It continues similarly if the maximum fuzz factor is larger.

The '`-F` *`lines`*' or'`--fuzz=`*`lines`*' option sets the maximum fuzz factor to *`lines`*. This option only applies to context and unified diffs; it ignores up to *`lines`* lines while looking for the place to install a hunk. Note that a larger fuzz factor increases the odds of making a faulty patch. The default fuzz factor is 2; it may not be set to more than the number of lines of context in the diff, ordinarily 3.

If patch cannot find a place to install a hunk of the patch, it writes the hunk out to a reject file (see "Reject File Names" on page 188 for information on how reject files are named). It writes out rejected hunks in context format no matter what form the input patch is in. If the input is a normal or ed diff, many of the contexts are simply null. The line numbers on the hunks in the reject file may be different from those in the patch file; they show the approximate location where patch thinks the failed hunks belong in the new file rather than in the old one.

As it completes each hunk, patch tells you whether the hunk succeeded or failed, and if it failed, on which line (in the new file) patch thinks the hunk should go. If this is different from the line number specified in the diff, it tells you the offset. A single large offset may indicate that patch installed a hunk in the wrong place. patch also tells you if it used a fuzz factor to make the match, in which case you should also be slightly suspicious.

patch cannot tell if the line numbers are off in an ed script, and can only detect wrong line numbers in a normal diff when it finds a change or delete command. It may have the same problem with a context diff using a fuzz factor equal to or greater than the number of lines of context shown in the diff (typically 3). In these cases, you should probably look at a context diff between your original and patched input files to see if the changes make sense. Compiling without errors is a pretty good indication that the patch worked, but not a guarantee.

patch usually produces the correct results, even when it must make many guesses. However, the results are guaranteed only when the patch is applied to an exact copy of the file that the patch was generated from.

Removing empty files

Sometimes when comparing two directories, the first directory con-tains a file that the second directory does not. If you give diff the '`-N`' or '`--new-file`' option, it outputs a diff that deletes the contents of this file. By default, `patch` leaves an empty file after applying such a diff. The '`-E`' or '`--remove-empty-files`' option to `patch` deletes output files that are empty after applying the diff.

Multiple patches in a file

If the patch file contains more than one patch, patch tries to apply each of them as if they came from separate patch files. This means that it determines the name of the file to patch for each patch, and that it examines the leading text before each patch for file names and prerequisite revision level (see “Tips for making patch distributions” on page 167 for more on that topic). For the second and subsequent patches in the patch file, you can give options and another original file name by separating their argument lists with a ‘+’. However, the argument list for a second or subsequent patch may not specify a new patch file, since that does not make sense. For example, to tell `patch` to strip the first three slashes from the name of the first patch in the patch file and none from subsequent patches, and to use ‘`code.c`’ as the first input file, you can use: `patch -p3 code.c + -p0 < patchfile`.

The ‘`-S`’ or ‘`--skip`’ option ignores the current patch from the `patch` file, but continue looking for the next patch in the file. Thus, to ignore the first and third patches in the patch file, you can use: `patch -S + + -S + < patch file`.

Messages and questions from patch

patch can produce a variety of messages, especially if it has trouble decoding its input. In a few situations where it's not sure how to proceed, patch normally prompts you for more information from the keyboard. There are options to suppress printing non-fatal messages and stopping for keyboard input. The message 'Hmm...' indicates that patch is reading text in the patch file, attempting to determine whether there is a patch in that text, and if so, what kind of patch it is. You can inhibit all terminal output from patch, unless an error occurs, by using the '-s', '--quiet', or '--silent' option. There are two ways you can prevent patch from asking you any questions. The '-f' or'--force' option assumes that you know what you are doing. It assumes the following:

- skip patches that do not contain file names in their headers;
- patch files even though they have the wrong version for the 'Prereq:' line in the patch;
- assume that patches are not reversed even if they look like they are.

The '-t' or '--batch' option is similar to '-f', in that it suppresses questions, but it makes somewhat different assumptions:

- skip patches that do not contain file names in their headers (the same as '-f');
- skip patches for which the file has the wrong version for the 'Prereq:' line in the patch;
- assume that patches are reversed if they look like they are.

  patch exits with a non-zero status if it creates any reject files. When applying a set of patches in a loop, you should check the exit status, so you don't apply a later patch to a partially patched file.

11

Tips for making patch distributions

The following discussions detail some things you should keep in mind if you are going to distribute patches for updating a software package.

Make sure you have specified the file names correctly, either in a context `diff` header or with an ‘`Index:`’ line. If you are patching files in a subdirectory, be sure to tell the patch user to specify a ‘`-p`’ or ‘`--strip`’ option as needed. Take care to not send out reversed patches, since these make people wonder whether they have already applied the patch.

To save people from partially applying a patch before other patches that should have gone before it, you can make the first patch in the patch file update a file with a name like ‘`patchlevel.h`’ or‘`version.c`’, which contains a patch level or version number. If the input file contains the wrong version number, patch will complain immediately.

An even clearer way to prevent this problem is to put a ‘`Prereq:`’ line before the patch. If the leading text in the patch file contains a line that starts with ‘`Prereq:`’, `patch` takes the next word from that line (normally a version number) and checks whether the next input file contains that word, preceded and followed by either white space or a newline. If not, `patch` prompts you for confirmation before proceeding. This makes it difficult to accidentally apply patches in the wrong order.

Since `patch` does not handle incomplete lines properly, make sure that all the source

files in your program end with a newline whenever you release a version.

To create a patch that changes an older version of a package into a newer version, first make a copy of the older version in a scratch directory. Typically you do that by unpacking a `tar` or `shar` archive of the older version.

You might be able to reduce the size of the patch by renaming or removing some files before making the patch. If the older version of the package contains any files that the newer version does not, or if any files have been renamed between the two versions, make a list of `rm` and `mv` commands for the user to execute in the old version directory before applying the patch. Then run those commands yourself in the scratch directory.

If there are any files that you don't need to include in the patch because they can easily be rebuilt from other files (for example, '`TAGS`' and output from `yacc` and `makeinfo`), replace the versions in the scratch directory with the newer versions, using `rm` and `ln` or `cp`.

Now you can create the patch. The de-facto standard `diff` format for patch distributions is context format with two lines of context, produced by giving `diff` the '`-C 2`' option. Do not use less than two lines of context, because patch typically needs at least two lines for proper operation.

Give diff the '`-P`' option in case the newer version of the package contains any files that the older one does not. Make sure to specify the scratch directory first and the newer directory second.

Add to the top of the patch a note telling the user any `rm` and `mv` commands to run before applying the patch. Then you can remove the scratch directory.

12

Invoking cmp

The `cmp` command compares two files, and if they differ, tells the first byte and line number where they differ.

Its arguments are: `cmp` *`options`* `...` *`from-file`* [*`to-file`*].

The file name '`-`' is always the standard input. `cmp` also uses the standard input if one file name is omitted.

An exit status of 0 means no differences were found, 1 means some differences were found, and 2 means trouble.

Options to cmp

The following is a summary of all of the options that GNU cmp accepts. Most options have two equivalent names, one of which is a single letter preceded by '-', and the other of which is a long name preceded by '--'. Multiple single letter options (unless they take an argument) can be combined into a single command line word: '-cl' is equivalent to '-c -l'.

'-c'
: Print the differing characters. Display control characters as a '^' followed by a letter of the alphabet and precede characters that have the high bit set with 'M-' (which stands for "meta").

'--ignore-initial=*bytes*'
: Ignore any differences in the the first *bytes* bytes of the input files. Treat files with fewer than *bytes* bytes as if they are empty.

'-l'
: Print the (decimal) offsets and (octal) values of all differing bytes.

'--print-chars'
: Print the differing characters. Display control characters as a '^' followed by a letter of the alphabet and precede characters that have the high bit set with 'M-' (which stands for "meta").

'--quiet'
'-s'
'--silent'
: Do not print anything; return exit status indicating whether files differ.

'--verbose'
: Print the (decimal) offsets and (octal) values of all differing bytes.

'-v'
'--version'
: Output the version number of cmp.

13

Invoking `diff`

`diff` *`options`* `...` *`from-file to-file`* is the format for running the `diff` command.

In the simplest case, `diff` compares the contents of the two files *`from-file`* and *`to-file`*. A file name of '`-`' stands for text read from the standard input. As a special case, '`diff - -`' compares a copy of standard input to itself. If *`from-file`* is a directory and *`to-file`* is not, diff compares the file in *`from-file`* whose file name is that of *`to-file`*, and vice versa. The non-directory file must not be '`-`'. If both *`from-file`* and *`to-file`* are directories, `diff` compares corresponding files in both directories, in alphabetical order; this comparison is not recursive unless the '`-r`' or '`--recursive`' option is given. `diff` never compares the actual contents of a directory as if it were a file. The file that is fully specified may not be standard input, because standard input is nameless and the notion of "file with the same name" does not apply.

`diff` options begin with '`-`', so normally *`from-file`* and *`to-file`* may not begin with '`-`'. However, '`--`' as an argument by itself treats the remaining arguments as file names even if they begin with '`-`'.

An exit status of 0 means no differences were found, 1 means some differences were found, and 2 means trouble.

Options to diff

The following is a summary of all of the options that GNU diff accepts. Most options have two equivalent names, one of which is a single letter preceded by '-', and the other of which is a long name preceded by '--'. Multiple single letter options (unless they take an argument) can be combined into a single command line word: '-ac' is equivalent to '-a -c'. Long named options can be abbreviated to any unique prefix of their name. Brackets ([and]) indicate that an option takes an optional argument.

'-*lines*'
: Show *lines* (an integer) lines of context. This option does not specify an output format by itself; it has no effect unless it is combined with '-c' (see "Context format" on page 113) or '-u' (see "Unified format" on page 115). This option is obsolete. For proper operation, patch typically needs at least two lines of context.

'-a'
: Treat all files as text and compare them line-by-line, even if they do not seem to be text. See "Binary files and forcing text comparisons" on page 107.

'-b'
: Ignore changes in amount of white space. See "Suppressing differences in blank and tab spacing" on page 102.

'-B'
: Ignore changes that just insert or delete blank lines. See "Suppressing differences in blank lines" on page 103.

'--binary'
: Read and write data in binary mode. See "Binary files and forcing text comparisons" on page 107.

'--brief'
: Report only whether the files differ, not the details of the differences. See "Summarizing which files differ" on page 106.

'-c'
: Use the context output format. See "Context format" on page 113.

'-C *lines*'

'--context[=*lines*]'
: Use the context output format, showing *lines* (an integer) lines of context, or three if *lines* is not given. See "Context format" on page 113. For proper operation, patch typically needs at least two lines of context.

'--changed-group-format=*format*'
: Use *format* to output a line group containing differing lines from both files in if-then-else format. See "Line group formats" on page 124.

‘-d’
Change the algorithm perhaps find a smaller set of changes. This makes diff slower (sometimes much slower). See “diff performance tradeoffs” on page 137.

‘-D *name*’
Make merged ‘#ifdef’ format output, conditional on the pre-processor macro *name*. See “Merging files with if-then-else” on page 124.

‘-e’
‘--ed’
Make output that is a valid ed script. See “ed scripts” on page 120.

‘--exclude= *pattern*’
When comparing directories, ignore files and subdirectories whose basenames match *pattern*. See “Comparing directories” on page 131.

‘--exclude-from= *file*’
When comparing directories, ignore files and subdirectories whose basenames match any pattern contained in *file*. See “Comparing directories” on page 131.

‘--expand-tabs’
Expand tabs to spaces in the output, to preserve the align-ment of tabs in the input files. See “Preserving tabstop alignment” on page 134.

‘-f’
Make output that looks vaguely like an ed script but has changes in the order they appear in the file. See “Forward ed scripts” on page 122.

‘-F *regexp*’
In context and unified format, for each hunk of differences, show some of the last preceding line that matches *regexp*. See “Suppressing lines matching a regular expression” on page 105.

‘--forward-ed’
Make output that looks vaguely like an ed script but has changes in the order they appear in the file. See “Forward ed scripts” on page 122.

‘-h’
This option currently has no effect; it is present for Unix compatibility.

‘-H’
Use heuristics to speed handling of large files that have nu-merous scattered small changes. See “diff performance tradeoffs” on page 137.

‘--horizon-lines= *lines*’
Do not discard the last *lines* lines of the common prefix and the first *lines* lines of the common suffix. See “diff performance tradeoffs” on page 137.

‘`-i`’
Ignore changes in case; consider uppercase and lowercase letters equivalent. See “Suppressing case differences” on page 104.

‘`-I` *`regexp`*’
Ignore changes that just insert or delete lines that match *`regexp`*. See “Suppressing lines matching a regular expression” on page 105.

‘`--ifdef=`*`name`*’
Make merged if-then-else output using *`name`*. See “Merging files with if-then-else” on page 124.

‘`--ignore-all-space`’
Ignore white space when comparing lines. See “Suppressing differences in blank and tab spacing” on page 102.

‘`--ignore-blank-lines`’
Ignore changes that just insert or delete blank lines. See “Suppressing differences in blank lines” on page 103.

‘`--ignore-case`’
Ignore changes in case; consider upper- and lower-case to be the same. See “Suppressing case differences” on page 104.

‘`--ignore-matching-lines=`*`regexp`*’
Ignore changes that just insert or delete lines that match *`regexp`*. See “Suppressing lines matching a regular expression” on page 105.

‘`--ignore-space-change`’
Ignore changes in amount of white space. See “Suppressing differences in blank and tab spacing” on page 102.

‘`--initial-tab`’
Output a tab rather than a space before the text of a line in normal or context format. This causes the alignment of tabs in the line to look normal. See “Preserving tabstop alignment” on page 134

‘`-l`’
Pass the output through `pr` to paginate it. See “Paginating diff output” on page 135.

‘`-L` *`label`*’
Use *`label`* instead of the file name in the context format (see “Detailed description of context format” on page 113) and unified format (see “Detailed description of unified format” on page 115) headers. See “RCS scripts” on page 122.

‘`--label=`*`label`*’
Use *`label`* instead of the file name in the context format (see “Detailed description of context format” on page 113) and unified format (see “Detailed description of unified format” on page 115) headers.

‘`--left-column`’
Print only the left column of two common lines in side by side format. See “Controlling side by side format” on page 120.

‘`--line-format=`*`format`*’
Use *`format`* to output all input lines in if-then-else format. See “Line formats” on page 127.

‘`--minimal`’
Change the algorithm to perhaps find a smaller set of changes. This makes `diff` slower (sometimes much slower). See “diff performance tradeoffs” on page 137.

‘`-n`’
Output RCS-format diffs; like ‘`-f`’ except that each command specifies the number of lines affected. See“RCS scripts” on page 122.

‘`-N`’
‘`--new-file`’
In directory comparison, if a file is found in only one directory, treat it as present but empty in the other directory. See “Comparing directories” on page 131.

‘`--new-group-format=`*`format`*’
Use *`format`* to output a group of lines taken from just the second file in `if-then-else` format. See “Line group formats” on page 124.

‘`--new-line-format=`*`format`*’
Use *`format`* to output a line taken from just the second file in `if-then-else` format. See “Line formats” on page 127.

‘`--old-group-format=`*`format`*’
Use *`format`* to output a group of lines taken from just the first file in if-then-else format. See “Line group formats” on page 124.

‘`--old-line-format=`*`format`*’
Use *`format`* to output a line taken from just the first file in if-then- else format. See “Line formats” on page 127.

‘`-p`’
Show which C function each change is in. See “Showing C function headings” on page 117.

‘`-P`’
When comparing directories, if a file appears only in the second directory of the two, treat it as present but empty in the other. See “Comparing directories” on page 131.

‘`--paginate`’
Pass the output through pr to paginate it. See “Paginating diff output” on page 135.

‘`-q`’
Report only whether the files differ, not the details of the differences. See “Summarizing which files differ” on page 106.

‘`-r`’
When comparing directories, recursively compare any sub-directories found. See “Comparing directories” on page 131.

‘`--rcs`’
Output RCS-format diffs; like ‘`-f`’ except that each command specifies the number of lines affected. See “RCS scripts” on page 122.

‘`--recursive`’
When comparing directories, recursively compare any sub-directories found. See “Comparing directories” on page 131.

‘`--report-identical-files`’
Report when two files are the same. See “Comparing directories” on page 131.

‘`-s`’
Report when two files are the same. See “Comparing directories” on page 131.

‘`-S` *`file`*’
When comparing directories, start with the file, *`file`*. This is used for resuming an aborted comparison. See “Comparing directories” on page 131.

‘`--sdiff-merge-assist`’
Print extra information to help `sdiff`. `sdiff` uses this option when it runs `diff`. This option is not intended for users to use directly.

‘`--show-c-function`’
Show which C function each change is in. See “Showing C function headings” on page 117.

‘`--show-function-line=`*`regexp`*’
In context and unified format, for each hunk of differences, show some of the last preceding line that matches *`regexp`*. See “Suppressing lines matching a regular expression” on page 105.

‘`--side-by-side`’
Use the side by side output format. See “Controlling side by side format” on page 120.

‘`--speed-large-files`’
Use heuristics to speed handling of large files that have nu-merous scattered small changes. See “diff performance tradeoffs” on page 137.

‘`--starting-file=`*`file`*’
When comparing directories, start with the file, *`file`*. This is used for resuming an aborted comparison. See “Comparing directories” on page 131.

‘`--suppress-common-lines`’
Do not print common lines in side by side format. See “Controlling side by side format” on page 120.

‘`-t`’
Expand tabs to spaces in the output, to preserve the alignment of tabs in the input files. See “Preserving tabstop alignment” on page 134.

‘`-T`’
Output a tab rather than a space before the text of a line in normal or context format. This causes the alignment of tabs in the line to look normal. See “Preserving tabstop alignment” on page 134.

‘`--text`’
Treat all files as text and compare them line-by-line, even if they do not appear to be text. See “Binary files and forcing text comparisons” on page 107.

‘`-u`’
Use the unified output format. See “Unified format” on page 115.

‘`--unchanged-group-format=`*`format`*’
Use *`format`* to output a group of common lines taken from both files in if-then-else format. See “Line group formats” on page 124.

‘`--unchanged-line-format=`*`format`*’
Use *`format`* to output a line common to both files in if-then-else format. See “Line formats” on page 127.

‘`--unidirectional-new-file`’
When comparing directories, if a file appears only in the second directory of the two, treat it as present but empty in the other. See “Comparing directories” on page 131.

‘`-U` *`lines`*’
‘`--unified[=` *`lines`*`]`’
Use the unified output format, showing *`lines`* (an integer) lines of context, or

three if *lines* is not given. See “Unified format” on page 115. For proper operation, patch typically needs at least two lines of context.

‘-v’

‘--version’
Output the version number of diff.

‘-w’
Ignore white space when comparing lines. See“Suppressing differences in blank and tab spacing” on page 102.

‘-W *columns*’

‘--width=*columns*’
Use an output width of *columns* in side by side format. See “Controlling side by side format” on page 120.

‘-x *pattern*’
When comparing directories, ignore files and subdirectories whose basenames match *pattern*. See“Comparing directories” on page 131.

‘-X *file*’
When comparing directories, ignore files and subdirectories whose basenames match any pattern contained in *file*. See “Comparing directories” on page 131.

‘-y’
Use the side by side output format. See “Controlling side by side format” on page 120.

14

Invoking `diff3`

The `diff3` command compares three files and outputs descriptions of their differences.

Its arguments are as follows: `diff3` *`options`* `...`*`mine older yours`*.

The files to compare are *`mine`*, *`older`*, and *`yours`*. At most one of these three file names may be '`-`', which tells `diff3` to read the standard input for that file. An exit status of 0 means `diff3` was successful, 1 means some conflicts were found, and 2 means trouble.

Options to diff3

The following is a summary of all of the options that GNU diff3 accepts. Multiple single letter options (unless they take an argument) can be combined into a single command line argument.

‘-a’
: Treat all files as text and compare them line-by-line, even if they do not appear to be text. See “Binary files and forcing text comparisons” on page 107.

‘-A’
: Incorporate all changes from older to yours into mine, sur-rounding all conflicts with bracket lines. See “Marking conflicts” on page 148.

‘-e’
: Generate an ed script that incorporates all the changes from older to yoursinto mine. See “Selecting which changes to incorporate” on page 147.

‘-E’
: Like ‘-e’, except bracket lines from overlapping changes’ first and third files. See “Marking conflicts” on page 148. With ‘-e’, an overlapping change looks like this:

```
<<<<<<< mine
lines from mine
=======
lines from yours
>>>>>>> yours
```

‘--ed’
: Generate an ed script that incorporates all the changes from older to yours into mine. See “Selecting which changes to incorporate” on page 147.

‘--easy-only’
: Like ‘-e’, except output only the non-overlapping changes. See “Selecting which changes to incorporate” on page 147.

‘-i’
: Generate ‘w’ and ‘q’ commands at the end of the ed script for System V compatibility. This option must be combined with one of the ‘-AeExX3’ options, and may not be combined with ‘-m’. See “Saving the changed file” on page 152.

‘--initial-tab’
: Output a tab rather than two spaces before the text of a line in normal format. This causes the alignment of tabs in the line to look normal. See “Preserving tabstop alignment” on page 134.

‘-L *label*’
‘--label=*label*’
Use the label, *label*, for the brackets output by the ‘-A’, ‘-E’ and ‘-X’ options. This option may be given up to three times, one for each input file. The default labels are the names of the input files. Thus ‘diff3 -L X -L Y -L Z -m A B C’ acts like ‘diff3 -m A B C’, except that the output looks like it came from files named ‘X’, ‘Y’ and ‘Z’ rather than from files named ‘A’, ‘B’ and ‘C’. See “Marking conflicts” on page 148.

‘-m’
‘--merge’
Apply the edit script to the first file and send the result to standard output. Unlike piping the output from diff3 to ed, this works even for binary files and incomplete lines. ‘-A’ is assumed if no edit script option is specified. See “Generating the merged output directly” on page 150.

‘--overlap-only’
Like ‘-e’, except output only the overlapping changes. See “Selecting which changes to incorporate” on page 147.

‘--show-all’
Incorporate all unmerged changes from *older* to *yours* into *mine*, surrounding all overlapping changes with bracket lines. See “Marking conflicts” on page 148.

‘--show-overlap’
Like ‘-e’, except bracket lines from overlapping changes’ first and third files. See “Marking conflicts” on page 148.

‘-T’
Output a tab rather than two spaces before the text of a line in normal format. This causes the alignment of tabs in the line to look normal. See “Preserving tabstop alignment” on page 134.

‘--text’
Treat all files as text and compare them line-by-line, even if they do not appear to be text. See “Binary files and forcing text comparisons” on page 107.

‘-v’
‘--version’
Output the version number of diff3.

‘-x’
Like ‘-e’, except output only the overlapping changes. See “Selecting which changes to incorporate” on page 147.

‘`-X`’
: Like ‘`-E`’, except output only the overlapping changes. In other words, like ‘`-x`’, except bracket changes as in ‘`-E`’. See “Marking conflicts” on page 148.

‘`-3`’
: Like ‘`-e`’, except output only the nonoverlapping changes. See “Selecting which changes to incorporate” on page 147.

15

Invoking `patch`

Normally `patch` is invoked using: `patch` < patchfile.

The full format for invoking `patch` is:

`patch` *`options`*`...[`*`origfile`* `[`*`patchfile`*`]] [+` *`options`* `...[`*`origfile`*`]]...`

If you do not specify *`patchfile`*,or if *`patchfile`* is '`-`', `patch` reads the patch (that is, the `diff` output) from the standard input.

You can specify one or more of the original files as *`orig`* arguments; each one and options for interpreting it is separated from the others with a '`+`'. See "Multiple patches in a file" on page 164 for more information.

If you do not specify an input file on the command line, patch tries to figure out from the leading text (any text in the patch that comes before the diff output) which file to edit. In the header of a context or unified diff, patch looks in lines beginning with '`***`', '`---`', or '`+++`'; among those, it chooses the shortest name of an existing file. Otherwise, if there is an '`Index:`' line in the leading text, `patch` tries to use the file name from that line. If `patch` cannot figure out the name of an existing file from the leading text, it prompts you for the name of the file to patch.

If the input file does not exist or is read-only, and a suitable RCS or SCCS file exists, patch attempts to check out or get the file before proceeding. By default, `patch` replaces the original input file with the patched version, after renaming the original

file into a backup file (see “Backup File Names” on page Backup File Names for a description of how patch names backup files). You can also specify where to put the output with the ‘`-o output-file`’ or ‘`--output=` *`output-file`*’ option.

Applying Patches in Other Directories

The ‘`-d` *`directory`*’ or ‘`--directory=`*`directory`*’ option to `patch` makes directory directory the current directory for interpreting both file names in the patch file, and file names given as arguments to other options (such as ‘`-B`’ and ‘`-o`’). For example, while in a news reading program, you can patch a file in the ‘`/usr/src/emacs`’ directory directly from the article containing the patch like the following example:

```
| patch -d /usr/src/emacs
```

Sometimes the file names given in a patch contain leading directories, but you keep your files in a directory different from the one given in the patch. In those cases, you can use the ‘`-p[`*`number`*`]`’ or ‘`--strip[=`*`number`*`]`’ option to set the file name strip count to *`number`*. The `strip` count tells `patch` how many slashes, along with the directory names between them, to strip from the front of file names. ‘`-p`’ with no numbergiven is equivalent to ‘`-p0`’. By default, `patch` strips off all leading directories, leaving just the base file names, except that when a file name given in the patch is a relative file name and all of its leading directories already exist, `patch` does not strip off the leading directory. (A relative file name is one that does not start with a slash.)

`patch` looks for each file (after any slashes have been stripped) in the current directory, or if you used the ‘`-d` directory’ option, in that directory. For example, suppose the file name in the patch file is ‘`/gnu/src/emacs/etc/new`’. Using ‘`-p`’ or ‘`-p0`’ gives the entire file name unmodified, ‘`-p1`’ gives ‘`gnu/src/emacs/etc/new`’ (no leading slash), ‘`-p4`’ gives ‘`etc/news`’, and not specifying ‘`-p`’ at all gives ‘`news`’.

Backup File Names

Normally, patch renames an original input file into a backup file by appending to its name the extension '`.orig`', or '`~`' on systems that do not support long file names. The '`-b backup-suffix`' or '`--suffix=`*`backup-suffix`*' option uses *`backup-suffix`* as the backup extension instead. Alternately, you can specify the extension for backup files with the `SIMPLE_BACKUP_SUFFIX` environment variable, which the options over-ride.

`patch` can also create numbered backup files the way GNU Emacs does. With this method, instead of having a single backup of each file, patch makes a new backup file name each time it patches a file. For example, the backups of a file named '`sink`' would be called, successively, '`sink.~1~`', '`sink.~2~`', '`sink.~3~`', etc. The '`-V` *`backup-style`*' or '`--version-control=` *`backup-style`*' option takes as an argument a method for creating backup file names. You can alternately control the type of backups that patch makes with the `VERSION_CONTROL` environment variable, which the '`-V`' option overrides. The value of the `VERSION_CONTROL` environment variable and the argu-ment to the '`-V`' option are like the GNU Emacs version-control variable (see "Transposing Text" in ***The GNU Emacs Manual***, for more informa-tion on backup versions in Emacs). They also recognize synonyms that are more descriptive. The valid values are listed in the following; unique abbreviations are acceptable.

'`t`'
'`numbered`'
: Always make numbered backups.

'`nil`'
'`existing`'
: Make numbered backups of files that already have them, simple backups of the others. This is the default.

'`never`'
'`simple`'
: Always make simple backups.

Alternately, you can tell patch to prepend a prefix, such as a directory name, to produce backup file names.

The '`-B` *`backup-prefix`*' or '`--prefix=`*`backup-prefix`*' option makes backup files by prepending backup-prefix to them. If you use this option, patch ignores any '`-b`' option that you give.

If the backup file already exists, patch creates a new backup file name by changing the first lowercase letter in the last component of the file name into uppercase. If there are

no more lowercase letters in the name, it removes the first character from the name. It repeats this process until it comes up with a backup file name that does not already exist.

If you specify the output file with the ‘`-o`’ option, that file is the one that is backed up, not the input file.

Reject File Names

The names for reject files (files containing patches that `patch` could not find a place to apply) are normally the name of the output file with '`.rej`' appended (or '`#`' on systems that do not support long file names).

Alternatively, you can tell patch to place all of the rejected patches in a single file. The '`-r` *`reject-file`*' or'`--reject-file=` *`reject-file`*' option uses *`reject-file`* as the reject file name.

Options to patch

The following summarizes the options that patch accepts. Older versions of patch do not accept long-named options or the '-t', '-E', or '-V' options.

Multiple single-letter options that do not take an argument can be combined into a single command line argument (with only one dash). Brackets ([and]) indicate that an option takes an optional argument.

'-b *backup-suffix*'
: Use *backup-suffix* as the backup extension instead of '.orig' or '~'. See "Backup File Names" on page 186.

'-B *backup-prefix*'
: Use *backup-prefix* as a prefix to the backup file name. If this option is specified, any '-b' option is ignored. See "Backup File Names" on page 186.

'--batch'
: Do not ask any questions. See "Messages and questions from patch" on page 165.

'-c'
'--context'
: Interpret the patch file as a context diff. See "Selecting the patch input format" on page 159.

'-d *directory*'
'--directory= *directory*'
: Makes directory *directory* the current directory for interpreting both file names in the patch file, and file names given as arguments to other options. See "Applying Patches in Other Directories" on page 185.

'-D *name*'
: Make merged if-then-else output using format. See "Merging files with if-then-else" on page 124.

'--debug=*number*'
: Set internal debugging flags. Of interest only to patch patchers.

'-e'
'--ed'
: Interpret the patch file as an ed script. See "Selecting the patch input format" on page 159.

'-E'
: Remove output files that are empty after the patches have been applied. See "Removing empty files" on page 163.

‘-f’
Assume that the user knows exactly what he or she is doing, and ask no questions. See “Messages and questions from patch” on page 165.

‘-F *lines*’
Set the maximum fuzz factor to *lines*. See “Helping patch find inexact matches” on page 161.

‘--force’
Assume that the user knows exactly what he or she is doing, and ask no questions. See “Messages and questions from patch” on page 165.

‘--forward’
Ignore patches that patch thinks are reversed or already applied. See also ‘-R’. See “Applying reversed patches” on page 160.

‘--fuzz=*lines*’
Set the maximum fuzz factor to *lines*. See “Helping patch find inexact matches” on page 161.

‘--help’
Print a summary of the options that patch recognizes, then exit.

‘--ifdef= *name*’
Make merged if-then-else output using format. See “Merging files with if-then-else” on page 124.

‘--ignore-white-space’
‘-l’
Let any sequence of white space in the patch file match any sequence of white space in the input file. See “Applying patches with changed white space” on page 160.

‘-n’
‘--normal’
Interpret the patch file as a normal diff. See “Selecting the patch input format” on page 159.

‘-N’
Ignore patches that patch thinks are reversed or already applied. See also ‘-R’. See “Applying reversed patches” on page 160.

‘-o *output-file*’
‘--output=*output-file*’
Use *output-file* as the output file name.

‘-p[*number*]’
Set the file name strip count to *number*. See “Applying Patches in Other Directories” on page 185.

‘--prefix=*backup-prefix*’
Use *backup-prefix* as a prefix to the backup file name. If this option is specified, any ‘-b’ option is ignored. See “Backup File Names” on page 186.

‘--quiet’
Work silently unless an error occurs. See “Messages and questions from patch” on page 165.

‘-r *reject-file*’
Use *reject-file* as the reject file name. See “Reject File Names” on page 188.

‘-R’
Assume that this patch was created with the old and new files swapped. See “Applying reversed patches” on page 160.

‘--reject-file=reject-file’
Use reject-file as the reject file name. See “Reject File Names” on page 188.

‘--remove-empty-files’
Remove output files that are empty after the patches have been applied. See “Removing empty files” on page 163.

‘--reverse’
Assume that this patch was created with the old and new files swapped. See “Applying reversed patches” on page 160.

‘-s’
Work silently unless an error occurs. See “Messages and questions from patch” on page 165.

‘-S’
Ignore this patch from the patch file, but continue looking for the next patch in the file. See “Multiple patches in a file” on page 164.

‘--silent’
Work silently unless an error occurs. See “Messages and questions from patch” on page 165.

‘--skip’
Ignore this patch from the patch file, but continue looking for the next patch in the file. See “Multiple patches in a file” on page 164.

‘--strip[=*number*]’
Set the file name strip count to *number*. See “Applying Patches in Other Directories” on page 185.

‘--suffix=*backup*-suffix’
Use *backup-suffix* as the backup extension instead of ‘.orig’ or ‘~’. “Backup File Names” on page 186.

‘-t’
Do not ask any questions. See “Messages and questions from patch” on page 165.

‘-u’
‘--unified’
Interpret the patch file as a unified diff. See “Selecting the patch input format” on page 159.

‘-v’
Output the revision header and patch level of patch.

‘-V *backup-style*’
Select the kind of backups to make. See “Backup File Names” on page 186.

‘--version’
Output the revision header and patch level of patch, then exit.

‘--version=control=*backup-style*’
Select the kind of backups to make. See “Backup File Names” on page 186.

‘-x *number*’
Set internal debugging flags. Of interest only to patch patchers.

16

Invoking `sdiff`

The `sdiff` command merges two files and interactively outputs the results.

Its arguments are: `sdiff -o` *`outfile options ... from-file to-file`*.

This merges *`from-file`* with *`to-file`*, with output to *`outfile`*.If *`from-file`* is a directory and *`to-file`* is not, sdiff compares the file in from-filewhose file name is that of `to-file`, and vice versa. *`from-file`* and *`to-file`* may not both be directories.

`sdiff` options begin with '`-`', so normally *`from-file`* and *`to-file`* may not begin with '`-`'. However, '`--`' as an argument by itself treats the remaining arguments as file names even if they begin with '`-`'. You may not use '`-`' as an input file. An exit status of 0 means no differences were found, 1 means some differences were found, and 2 means trouble.

`sdiff` without '`-o`' (or '`--output`') produces a side-by-side difference. This usage is obsolete; use '`diff --side-by-side`' instead.

Options to sdiff

The following is a summary of all of the options that GNU sdiff accepts. Each option has two equivalent names, one of which is a single letter preceded by '-', and the other of which is a long name preceded by '--'. Multiple single letter options (unless they take an argument) can be combined into a single command line argument. Long named options can be abbreviated to any unique prefix of their name.

'`-a`'
Treat all files as text and compare them line-by-line, even if they do not appear to be text. See "Binary files and forcing text comparisons" on page 107.

'`-b`'
Ignore changes in amount of whitespace. See "Suppressing differences in blank and tab spacing" on page 102.

'`-B`'
Ignore changes that just insert or delete blank lines. See "Suppressing differences in blank lines" on page 103.

'`-d`'
Change the algorithm to perhaps find a smaller set of changes. This makes `sdiff` slower (sometimes much slower). See "diff performance tradeoffs" on page 137.

'`-H`'
Use heuristics to speed handling of large files that have numerous scattered small changes. See "diff performance tradeoffs" on page 137.

'`--expand-tabs`'
Expand tabs to spaces in the output, to preserve the alignment of tabs in the input files. See "Preserving tabstop alignment" on page 134.

'`-i`'
Ignore changes in case; consider uppercase and lowercase to be the same. See "Suppressing case differences" on page 104.

'`-I` *`regexp`*'
Ignore changes that just insert or delete lines that match *`regexp`*. See "Suppressing lines matching a regular expression" on page 105.'`--ignore-all-space`'
Ignore white space when comparing lines. See "Suppressing differences in blank and tab spacing" on page 102.

'`--ignore-blank-lines`'
Ignore changes that just insert or delete blank lines. See"Suppressing differences in blank lines" on page 103.

‘`--ignore-case`’
Ignore changes in case; consider uppercase and lowercase to be the same. See “Suppressing case differences” on page 104.

‘`--ignore-matching-lines=`*`regexp`*’
Ignore changes that just insert or delete lines that match *`regexp`*. See “Suppressing lines matching a regular expression” on page 105.

‘`--ignore-space-change`’
Ignore changes in amount of whitespace. See “Suppressing differences in blank and tab spacing” on page 102.

‘`-l`’
‘`--left-column`’
Print only the left column of two common lines. See “Controlling side by side format” on page 120.

‘`--minimal`’
Change the algorithm to perhaps find a smaller set of changes. This makes `sdiff` slower (sometimes much slower). See “diff performance tradeoffs” on page 137.

‘`-o` *`file`*’
‘`--output=`*`file`*’
Put merged output into *`file`*. This option is required for merging.

‘`-s`’
‘`--suppress-common-lines`’
Do not print common lines. See “Controlling side by side format” on page 120.

‘`--speed-large-files`’
Use heuristics to speed handling of large files that have numerous scattered small changes. See “diff performance tradeoffs” on page 137.

‘`-t`’
Expand tabs to spaces in the output, to preserve the alignment of tabs in the input files. See “Preserving tabstop alignment” on page 134.

‘`--text`’
Treat all files as text and compare them line-by-line, even if they do not appear to be text. See “Binary files and forcing text comparisons” on page 107.

‘`-v`’
‘`--version`’
Output the version number of `sdiff`.

‘`-w` *`columns`*’
‘`--width=`*`columns`*’
Use an output width of *`columns`*. See “Controlling side by side format” on page 120. For historical reasons, this option is ‘`-W`’ `indiff`,‘`-w`’ in `sdiff`.

‘`-W`’
Ignore horizontal white space when comparing lines. See “Suppressing differences in blank and tab spacing” on page 102. For historical reasons, this option is ‘`-w`’ in `diff`,‘`-W`’ in `sdiff`.

17

Incomplete lines

When an input file ends in a non-newline character, its last line is called an incomplete line because its last character is not a newline. All other lines are called full lines and end in a newline character. Incomplete lines do not match full lines unless differences in white space are ignored (see “Suppressing differences in blank and tab spacing” on page 102).

An incomplete line is normally distinguished on output from a full line by a following line that starts with ‘`\`’. However, the RCS format (see “RCS scripts” on page 122) outputs the incomplete line as-is, without any trailing newline or following line. The side by side format normally represents incomplete lines as-is, but in some cases uses a ‘\’ or ‘/’ gutter marker; See “Controlling side by side format” on page 120. The if-then-else line format preserves a line’s incompleteness with ‘`%L`’, and discards the new-line with ‘`%l`’; see “Line formats” on page 127. Finally, with the ed and forward ed output formats (see “diff output formats” on page 109) diff cannot represent an incomplete line, so it pretends there was a newline and reports an error. For example, suppose ‘`f`’ and ‘`g`’ are one-byte files that contain just ‘`f`’ and ‘`g`’, respectively. Then ‘`diff f g`’ outputs

```
1c1
< f
\ No newline at end of file
```

```
---
> g
\ No newline at end of file
```

(The exact message may differ in non-English locales.) ‘`diff -n f g`’ outputs the following without a trailing newline:

```
d1 1
a1 1
g
```

‘`diff -e f g`’ reports two errors and outputs the following:

```
1c
g
.
```

18

Future projects

The following discussions have some ideas for improving GNU `diff` and `patch`. The GNU project has identified some improvements as potential programming projects for volunteers. You can also help by reporting any bugs that you find. If you are a programmer and would like to contribute something to the GNU project, please consider volunteering for one of these projects. If you are seriously contemplating work, please write to ‘`gnu@prep.ai.mit.edu`’ to coordinate with other volunteers.

Suggested projects for improving GNU diff and patch

One should be able to use GNU diff to generate a patch from any pair of directory trees, and given the patch and a copy of one such tree, use patch to generate a faithful copy of the other. Unfortunately, some changes to directory trees cannot be expressed using current patch formats; also, patch does not handle some of the existing formats. These shortcomings motivate the following suggested projects.

Handling changes to the directory structure

diff and patch do not handle some changes to directory structure. For example, suppose one directory tree contains a directory named 'D' with some subsidiary files, and another contains a file with the same name 'D'. 'diff -r' does not output enough information for patch to transform the the directory subtree into the file. There should be a way to specify that a file has been deleted without having to include its entire contents in the patch file. There should also be a way to tell patch that a file was renamed, even if there is no way for diff to generate such information. These problems can be fixed by extending the diff output format to represent changes in directory structure, and extending patch to understand these extensions.

Files that are neither directories nor regular files

Some files are neither directories nor regular files: they are unusual files like symbolic links, device special files, named pipes, and sockets. Currently, diff treats symbolic links like regular files; it treats other special files like regular files if they are specified at the top level, but simply reports their presence when comparing directories. This means that patch cannot represent changes to such files. For example, if you change which file a symbolic link points to, diff outputs the difference between the two files, instead of the change to the symbolic link.

diff should optionally report changes to special files specially, and patch should be extended to understand these extensions.

File names that contain unusual characters

When a file name contains an unusual character like a newline or white space, 'diff -r' generates a patch that patch cannot parse. The problem is with format of diff output, not just with patch, because with odd enough file names one can cause diff to generate a patch that is syntactically correct but patches the wrong files. The format of diff output should be extended to handle all possible file names.

Arbitrary limits

GNU `diff` can analyze files with arbitrarily long lines and files that end in incomplete lines. However, `patch` cannot patch such files. The `patch` internal limits on line lengths should be removed, and `patch` should be extended to parse `diff` reports of incomplete lines.

Handling files that do not fit in memory

`diff` operates by reading both files into memory. This method fails if the files are too large, and `diff` should have a fallback.

One way to do this is to scan the files sequentially to compute hash codes of the lines and put the lines in equivalence classes based only on hash code. Then compare the files normally. This does produce some false matches.

Then scan the two files sequentially again, checking each match to see whether it is real. When a match is not real, mark both the “matching” lines as changed. Then build an edit script as usual.

The output routines would have to be changed to scan the files se-quentially looking for the text to print.

Ignoring certain changes

It would be nice to have a feature for specifying two strings, one in from-file and one in to-file, which should be considered to match. Thus, if the two strings are ‘`foo`’ and ‘`bar`’, then if two lines differ only in that ‘`foo`’ in file 1 corresponds to ‘`bar`’ in file 2, the lines are treated as identical. It is not clear how general this feature can or should be, or what syntax should be used for it.

Reporting bugs

If you think you have found a bug in GNU `cmp`, `diff`, `diff3`, `sdiff`,or `patch`, report it by electronic mail to ‘`bug-gnu-utils@prep.ai.mit.edu`’. Send as precise a description of the problem as you can, including sample input files that produce the bug, if applicable. Because Larry Wall has not released a new version of `patch` since mid-1988 and the GNU version of `patch` has been changed since then, please send bug reports for patch by electronic mail to both ‘`bug-gnu-utils@prep.ai.mit.edu`’ and ‘`lwall@netlabs.com`’.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y