GNU Interactive Tools

A Set of Interactive Programs
Edition 2.1, for GIT version 4.3.7
July 1995

by Tudor Hulubei and Andrei Pitis
"Politehnica" University of Bucharest
Romania

1 Introduction

GIT is a set of interactive tools. It contains an extensible file system browser, an ascii/hex file viewer, a process viewer/killer and some other related utilities and shell scripts. It can be used to increase the speed and efficiency of most of the daily tasks as copying and moving files and directories, invoking editors, compressing and uncompressing files, creating and expanding archives, compiling programs, sending mail, etc. It looks nice, has colors (if the standard ANSI color sequences are supported) and is user-friendly.

GIT runs on a wide variety of UNIX systems because it uses the GNU Autoconf package to get system specific information. Please refer to the PLATFORMS file included in the standard distribution for a detailed list of systems on which GIT has been tested.

One of the main advantages of GIT is its flexibility. It is not limited to a given set of commands. The configuration file can be easily enhanced, allowing the user to add new commands or file operations, depending on its needs or preferences.

GIT also provides a shell like command prompt, just to make sure that the entire power of the UNIX shell commands is still there.

2 Distributing GNU Interactive Tools

GIT is "free software"; this means that everyone is free to use it and free to redistribute it on certain conditions. GIT is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GIT that they might get from you. The precise conditions are found in the GNU General Public License that comes with GIT and also appears following this section.

The easiest way to get a copy of GIT is from someone else who has it. You need not ask for our permission to do so, or tell any one else; just copy it. If you have access to the Internet, you can get the latest distribution version of GIT from host 'prep.ai.mit.edu' using anonymous login. See the file '/pub/gnu/GETTING.GNU.SOFTWARE' on that host to find out about your options for copying and which files to use.

You may also receive GNU Interactive Tools when you buy a computer. Computer manufacturers are free to distribute copies on the same terms that apply to everyone else. These terms require them to give you the full sources, including whatever changes they may have made, and to permit you to redistribute the GNU Interactive Tools received from them under the usual terms of the General Public License. In other words, the program must be free for you when you get it, not just free for the manufacturer.

3 Using GNU Interactive Tools

The GIT package contains three interactive programs and a few additional utilities. Here there is a description of each of them.

3.1 The GIT file system browser

`git` is a file system browser with some shell features designed to make your work much easier and much efficient. It displays one or two panels, each one containing a file system directory. You can browse the directory tree with the usual cursor keys, pressing *ENTER* when you want to enter or leave a directory and *TAB* when you want to change the panels.

Under the two panels there is a shell like input line which you can use to type normal shell commands. The input line can handle an unlimited number of characters and keeps a history of typed commands. `git` uses the GNU history library for that matter.

Under the input line there is a status bar. You can see there the status of the currently executed command, the warnings and errors and you will be prompted if a decision has to be taken.

3.1.1 Key binding conventions

`git` now provides a new, easy to remember, scheme to bind commands on keys. This is only a convention, if you define new key bindings you may, or may not follow it.

All the file commands start with *^C*. This prefix can be followed by some modifiers, in order to affect the default behavior of the given command. These modifiers are *b* and *r*.

b - this modifier specifies that the command will run in background:

```
^CM = CHMOD; chmod %s{New mode of %i: ,%m} %i;;;;y
```

defines a command that changes the current selected files mode in foreground, while

```
^CbM = B-CHMOD; chmod %s{New mode of %i: ,%m} %i&;;;;y
```

defines a background command that does the same thing.

r - this modifier specifies that the command will be run recursively:

```
^CrM = R-CHMOD; chmod -R %s{New mode of %i: ,} %i;;;;y
```

defines a command that recursively changes the mode of the selected files.

The *b* and *r* modifiers can be combined, the resulting command running recursively and in background:

```
^CbrM = B-R-CHMOD; chmod -R %s{New mode of %i: ,} %i&;;;;y
```

You should also note that for some commands (like `gzip`) there is no need for a non-recursive version. Running `gzip` recursively on files is harmless. If there is a directory between these files, `gzip` will recursively compress that directory, so you can use the same key binding for recursively and non-recursively compressing. In fact, it is a matter of selecting files or directories.

Other programs are less smarter than `gzip`. We can't run `chmod` recursively trying to change the mode of all the files in a directory to 644 because that directory might contain subdirectories and removing the execution permission from them is a bad idea. So, in this case, we need separate commands.

3.1.2 Panel modes

`git` has three major modes of displaying the panels. In the first (default) mode, two panels are displayed, each one using half of the screen. In the second mode, only one panel uses the entire screen. In the third mode, only the status bar, the input line and the clock are displayed, all the panels being hidden.

Briefly, a panel can use the entire screen or just half of it. Even when a panel is hidden, it still exists.

Users can switch between these three major modes as needed:

^X 0

Enlarges the other panel to use the entire screen. It also changes the minor mode to '`Display all`'. The current panel will become invisible ('`enlarge-other-panel`').

^X 1

Enlarges the current panel to use the entire screen. It also changes the minor mode to '`Display all`'. The other panel will become invisible ('`enlarge-panel`').

^X 2

Switches back to the two panel mode ('`two-panel-mode`').

^O, *ESC o*

Switches to the tty mode (no panels on the screen) ('`tty-mode`').

Basically a panel displays the files and directories in a directory. You can optionally specify some additional information about each entry (file, directory, ...) to be displayed (a minor mode).

When using the full screen mode, all the minor modes here can be used. In half screen mode, the '`panel-display-all`' mode is not available.

These are the panel minor modes:

ESC d o

Displays the entry owner and group ('`panel-display-owner-group`').

ESC d d

Displays the entry date and time ('`panel-display-date-time`').

ESC d s

Displays the entry size ('`panel-display-size`').

ESC d m

Displays the entry mode ('`panel-display-mode`').

ESC d f

Displays the entry full name ('`panel-display-full-name`').

ESC d a

Displays the entire information about file ('`panel-display-all`'). This mode is only available if the panel has been enlarged to use the entire screen with '`enlarge-panel`' or '`enlarge-other-panel`' ('`panel-display-all`').

There is also another way to change the panel minor modes:

^]

Switches to the next panel minor mode (‘`panel-display-next-mode`’).

3.1.3 Sorting methods

Entries in a panel can be sorted in different ways. These are the available options:

ESC s n

Displays the panel entries sorted by their names (‘`panel-sort-by-name`’).

ESC s e

Displays the panel entries sorted by their extensions (‘`panel-sort-by-extension`’).

ESC s s

Displays the panel entries sorted by their sizes (‘`panel-sort-by-size`’).

ESC s d

Displays the panel entries sorted by their ‘`last modified`’ stamps (‘`panel-sort-by-date`’).

ESC s m

Displays the panel entries sorted by their modes (‘`panel-sort-by-mode`’).

ESC s o i

Displays the panel entries sorted by their owner ids (‘`panel-sort-by-owner-id`’).

ESC s g i

Displays the panel entries sorted by their group ids (‘`panel-sort-by-group-id`’).

ESC s o n

Displays the panel entries sorted by their owner names (‘`panel-sort-by-owner-name`’).

ESC s g n

Displays the panel entries sorted by their group names (‘`panel-sort-by-group-name`’).

There is also another way to change the sort method:

ESC s u

Switches to the next panel sort method (‘`panel-sort-next-method`’).

3.1.4 Moving the cursor in the panel

Moving the cursor in the panel is very easy. If your keyboard has arrows, use them. If the arrow keys don’t work (it might be due to a badly configured *TERM* environment variable), you can use the Emacs commands bindings as well.

UP, *^P*

Moves the cursor vertically up one entry (‘`previous-line`’).

DOWN, *^N*

Moves the cursor vertically down one entry (‘`next-line`’).

HOME, *ESC <*

Moves the cursor on the first entry in the panel (‘`beginning-of-panel`’).

END, *ESC >*

Moves the cursor on the last entry in the panel (‘`end-of-panel`’).

PGUP, *ESC v*

Moves the cursor vertically down one page (‘`scroll-down`’).

PGDOWN, *^V*

Moves the cursor vertically down one page (‘`scroll-up`’).

^X P

In order to optimize the screen output, you can modify the scroll step (‘`set-scroll-step`’). This is the number of lines to try scrolling a panel when the cursor moves out. The ‘`StartupScrollStep`’ specifies the initial scroll step, but using ‘`set-scroll-step`’ you can dynamically change it.

TAB, *^I*, *^X o*

Moves the cursor in the other panel (‘`other-panel`’).

^X P

Switch the two panels. This command works even when `git` is not in the ‘`two panels`’ mode (‘`switch-panels`’).

3.1.5 Selecting files

INS, *^T*, *^X *, *^*

Toggle the ‘`selected`’ flag of the current file (‘`select-file`’).

^C s

Selects (marks) all the files matching a pattern (‘`select-files-matching-pattern`’). The user will be prompted for a pattern to match against.

^C u

Unselects (unmarks) all the files matching a pattern (‘`select-files-matching-pattern`’). The user will be prompted for a pattern to match against.

3.1.6 Incremental searching files in a panel

Users sometime need to search a file in a panel, especially when the panel contains a big number of entries. For that reason `git` provides an incremental search feature. Using forward and backward incremental search, files can be very easy located. Wrapped incremental search is also provided.

^S, *^Xs*

Incremental searches forward a file in the current panel (‘`isearch-forward`’). Pressing *^S* or *^Xs*

again will force `git` to go to the next entry that matches the current isearched string. When the end of the panel is reached, the isearch is restarted from its beginning.

^R, *^Xr*

Incremental searches backward a file in the current panel ('`isearch-backward`'). Pressing *^R* or *^Xr* again will force `git` to go to the next entry that matches the current isearched string. When the beginning of the panel is reached, the isearch is restarted from its end.

3.1.7 Using the input line

The input line is one of the main methods used by `git` to interact with the user. All the answers the user should give in order to perform some operation and all the shell like commands are built using it. So here is a description of all the basic editing operations that the '`input line`' provides. They are very much inspired from `Emacs`, so `Emacs` users should have no problem using them.

3.1.7.1 Inserting Text

Typing characters is the most usual way of inserting text into the input line. Key sequences starting with printable ascii characters are not allowed in `git` so typing `a` for example results in inserting `a` at the current point position. Of course, there are some other ways of inserting text into the command line and here there is a description of most of them.

ESC RET

Copies the current file name into the input line at the current point position ('`file-to-input-line`').

ESC ESC RET

Copies the other panel path into the input line at the current point position ('`other-path-to-input-line`').

^X ^G

Copies the names of all the selected files into the input line at the current point position ('`selected-files-to-input-line`').

3.1.7.2 Moving Point

^B, *LEFT*

Moves the point backward one character ('`backward-char`').

^F, *RIGHT*

Moves the point forward one character ('`forward-char`').

ESC b

Moves the point one word backward ('`backward-word`').

ESC f

Moves the point one word forward ('`forward-word`').

^A

Moves the cursor at the beginning of the input line ('`beginning-of-line`').

^E

Moves the cursor at the end of the input line (‘`end-of-line`’).

3.1.7.3 Deleting and killing text

DEL, *^D*

Deletes the character under the cursor (‘`delete-char`’).

^H, *BKSPC*

Deletes the character before the cursor (‘`backward-delete-char`’).

ESC BKSPC

Deletes backward one word (‘`backward-kill-word`’).

ESC k

Deletes the entire line (‘`kill-line`’).

^U

Deletes all the characters between the beginning of the input line and the point (‘`kill-to-beginning-of-line`’).

^K

Deletes all the characters between the point and the end of the input line (‘`kill-to-end-of-line`’).

ESC SPC

Deletes all the spaces around the point, leaving only one space (‘`just-one-space`’).

ESC \

Deletes all the spaces around the point (‘`delete-horizontal-space`’).

^W

Saves the region between the point and the mark into the kill "ring" and then kills it (‘`kill-region`’). Note that there is no real kill-ring here. The so-called kill-ring has only one entry.

ESC w

Saves the region between the point and the mark without killing it (‘`kill-ring-save`’).

3.1.7.4 Reusing recent input line arguments

A separate history is kept for both built-in and user-defined commands. If you call a command that you have used before, you can re-edit a previously entered string in order to minimize the amount of characters needed to be typed for the new one. There is no limit on the number of strings that can be kept in the history.

ESC p

Walks backward through the history of previously entered strings (‘`previous-history-element`’).

ESC n

Walks forward through the history of previously entered strings (‘`next-history-element`’).

3.1.7.5 Commands to set the mark

^SPC

Sets the mark at the current point position (‘`set-mark`’).

^X ^X

Exchange the current point position with the mark one (‘`exchange-point-and-mark`’).

3.1.7.6 Reinserting recently killed text

^Y

Reinserts a previously killed text at the current point position (‘`yank`’).

3.1.8 File operations

3.1.8.1 Copying Files

F5, *^C C*

Copies the currently selected files and directories to the user supplied path (‘`copy`’).

^C b C

Copies the currently selected files and directories to the user supplied path. The operation is performed in background (‘`B-COPY`’).

3.1.8.2 Moving Files

F6, *^C T*

Moves the currently selected files and directories to the user supplied path (‘`move`’).

^C b T

Moves the currently selected files and directories to the user supplied path. The operation is performed in background (‘`B-MOVE`’).

3.1.8.3 Creating Files

The easiest way to create a new file is to start an editor and put something in it. See Section 3.1.8.8 [Editing Files], page 11, for more information.

3.1.8.4 Deleting Files

F8, *^C D*

Deletes the currently selected files and directories (‘`delete`’).

^C b D

Deletes the currently selected files and directories. The operation is performed in background (‘`B-DELETE`’).

3.1.8.5 Linking Files

^C H

Creates a hard link from the current files to a user supplied file name (‘LINK’).

^C b H

Creates a hard link from the current files to a user supplied file name (‘B-LINK’). The action is performed in background.

^C S

Creates a symbolic link from the current files to a user supplied file name (‘SYMLINK’).

^C b S

Creates a symbolic link from the current files to a user supplied file name (‘B-SYMLINK’). The action is performed in background.

3.1.8.6 Renaming Files

^C R

Renames the current file or directory with the user supplied name (‘RENAME’).

^C b R

Renames the current file or directory with the user supplied name. The operation is performed in background (‘B-RENAME’).

3.1.8.7 Changing a file's inode mode, owner and group

^C M

Changes the mode of the currently selected files inodes (‘CHMOD’).

^C b M

Changes the mode of the currently selected files inodes. The operation is performed in background (‘B-CHMOD’).

^C r M

Recursively changes the modes of the selected files inodes if one of them is a directory (‘R-CHMOD’).

^C b r M

Recursively changes the modes of the selected files inodes if one of them is a directory. The operation is performed in background (‘B-R-CHMOD’).

^C O

Changes the owner of the currently selected files inodes (‘CHOWN’).

^C b O

Changes the owner of the currently selected files inodes. The operation is performed in background (‘B-CHOWN’).

^C r O

Recursively changes the owners of the selected files inodes if one of

them is a directory (‘R-CHOWN’).

^C b r O

Recursively changes the owners of the selected files inodes if one of them is a directory. The operation is performed in background (‘B-R-CHOWN’).

^C G

Changes the group of the currently selected files inodes (‘CHGRP’).

^C b G

Changes the group of the currently selected files inodes. The operation is performed in background (‘B-CHGRP’).

^C r G

Recursively changes the groups of the selected files inodes if one of them is a directory (‘R-CHGRP’).

^C b r G

Recursively changes the groups of the selected files inodes if one of them is a directory. The operation is performed in background (‘B-R-CHGRP’).

3.1.8.8 Editing Files

F4

Calls the default editor with the current file name as an argument (‘EDIT’).

^X e

Calls the default editor with the selected files names as arguments (‘MULTIPLE-EDIT’).

^X ^F

Creates a new file by calling the default editor with the user supplied file name as an argument (‘FILE-CREATE’).

^X 4 a

Calls the default editor in order to edit the `ChangeLog` file (‘CHANGE-LOG’).

The default editor can be specified using the *EDITOR* or *GIT_EDITOR* environment variables. See Section 4.1 [Environment Variables], page 23, for more information.

3.1.8.9 Viewing Files

F3, *^X h*

Calls the default viewer (`gitview`) with the current file name as argument (‘VIEW’).

^X v

Calls the default pager (`more`) with the currently selected file names as arguments (‘MULTIPLE-VIEW’).

3.1.8.10 Compressing Files

^C z

Compresses the currently selected files and directories (‘COMPRESS’).

^C b z

Compresses the currently selected files and directories. The operation is performed in background (‘B-COMPRESS’).

^C Z

Uncompresses the currently selected files and directories (‘UNCOMPRESS’).

^C b Z

Uncompresses the currently selected files and directories. The operation is performed in background (‘B-UNCOMPRESS’).

3.1.8.11 Encoding Files

^C e

Encodes the currently selected file (‘UUENCODE’).

^C b e

Encodes the currently selected file. The operation is performed in background (‘B-UUENCODE’).

^C E

Decodes the currently selected file (‘UUDECODE’).

^C b E

Decodes the currently selected file. The operation is performed in background (‘B-UUDECODE’).

3.1.8.12 Encrypting Files

^C p

Encrypts (using pgp) the current file (‘ENCRYPT’).

^C P

Decrypts (using pgp) the current file (‘DECRYPT’).

3.1.8.13 Comparing Files

^C =

Compares (using diff) the current ASCII file with the other panel current file (‘DIFF’). If both entries are directories, a recursive diff is performed.

^C ESC =

Compares (using diff) the current ASCII file with its latest backup. The latest backup is the file having the same name and a ’~’ at the end (‘LAST-BACKUP-DIFF’).

^C B

Compares the current file with the other panel current file. A binary comparison is performed (‘BINARY-COMPARE’).

3.1.8.14 Spelling Files

^X I

Runs the `ispell` command with the current file name as an argument.

3.1.8.15 Wiping Files

^C W

Calls `gitwipe` to wipe the selected files. Asks for confirmation before actually wiping them in order to avoid errors ('`WIPE`').

See Section 3.7 [gitwipe], page 21, for more information.

3.1.8.16 Searching Files

ESC %

Searches files on the file system, starting from the current directory ('`FIND`').

^X w

Locates the binary, source, and manual page files for a command ('`WHEREIS`').

^X W

Locate a command; display its pathname or alias ('`WHICH`').

3.1.8.17 Managing tar based archive files

^C a

Creates a `tar` archive containing all the currently selected files and directories ('`TAR`').

^C b a

Creates a `tar` archive containing all the currently selected files and directories. The operation is performed in background ('`B-TAR`').

^C A

Expands the `tar` archive pointed by the cursor into the current directory ('`UNTAR`').

^C b A

Expands the `tar` archive pointed by the cursor into the current directory. The operation is performed in background ('`B-UNTAR`').

^C x

Creates a compressed `tar` archive containing all the currently selected files and directories ('`TAR-COMPRESS`').

^C b x

Creates a compressed `tar` archive containing all the currently selected files and directories. The operation is performed in background ('`B-TAR-COMPRESS`').

^C X

Expands the compressed `tar` archive pointed by the cursor into the

current directory (‘UNCOMPRESS-UNTAR’).

^C b X

Expands the compressed `tar` archive pointed by the cursor into the current directory. The operation is performed in background (‘B-UNCOMPRESS-UNTAR’).

3.1.8.18 A different action for each file type

Many files on `UNIX` systems have one or more extensions specifying their types. For example, a file that ends in ‘`.c`’ is a file containing a `C` program, while a file ending in ‘`.tar.gz`’ is a `tar` archive compressed with the `gzip` utility. Having a default action for each file type, binded on the same key, seems to be a good idea because you can use that key to obtain type specific information about a file or to process it in some type specific way much easier. The `GIT` package contains a script called `gitaction` and a file name match utility called `gitmatch` that are used together to detect the current file type and perform a type specific action. See Section 3.9 [gitaction], page 21, for more information.

F2, *^X a*

Performs an action on the current file, depending on its type (‘FILE-ACTION’).

3.1.9 Directory operations

3.1.9.1 Creating directories

F7, *^X M*

Creates a new subdirectory in the current directory with the user supplied name (‘make-directory’).

3.1.9.2 Copying directories

F5 (for directories), *^C C (for directories)*

Copies the currently selected files and directories to the user supplied path (‘copy’).

^C b C (for directories)

Copies the currently selected files and directories to the user supplied path. The operation is performed in background (‘B-COPY’).

3.1.9.3 Deleting directories

F8 (for subdirectories), *^C D (for subdirectories)*

Deletes the currently selected files and directories (‘delete’).

^C b D (for directories)

Deletes the currently selected files and directories. The operation is performed in background (‘B-DELETE’).

3.1.9.4 Moving directories

F6 (for directories), *^C T (for directories)*

Moves the currently selected files and directories to the user supplied

path (‘move’).

^C b T (for directories)

Moves the currently selected files and directories to the user supplied path. The operation is performed in background (‘B-MOVE’).

3.1.9.5 Renaming directories

^C R (for directories)

Renames the current file or directory with the user supplied name (‘RENAME’).

^C b R (for directories)

Renames the current file or directory with the user supplied name. The operation is performed in background (‘B-RENAME’).

3.1.9.6 Summarize directory usage

^C U

Displays the output of the `du -s` command on the status line (‘DIRECTORY-USAGE’).

3.1.9.7 Changing directories

^X d, *^X ^D*

Changes the current working directory. The user is asked for a new directory name and the new directory is added to the directory history (‘change-directory’).

See Section 3.1.9.8 [Dirs History], page 15, for more information.

ESC c c

Change the current directory of the current panel to the directory of the other panel (‘conform-current-directory’).

ESC c o

Change the current directory of the other panel to the directory of the current panel (‘conform-other-directory’).

3.1.9.8 Directory History

Users usually work on a limited set of subdirectories. Providing a fast method of switching between a number of intensively used directories seems to be a good idea and `git` has a set of builtin commands for this.

Usually new directories are added to the directory history when the ‘change-directory’ built-in command is used. `git` also adds the current directory to the history list when started, when the directory history is reseted and when a command having a non empty ‘new-dir’ field successfully completes its execution. See Section 4.2.3.22 [new-dir], page 28, for more information.

^X ^N

Goes to the next directory in the history (‘next-directory’).

^X ^P

Goes to the previous directory in the history (‘`previous-directory`’).

^X ^R

Resets the entire directory history. As explained above, the current directory becomes the only directory in the history (‘`reset-directory-history`’).

3.1.9.9 Hot Keys

`git` provides default key bindings for switching to a number of important directories as "/", "..", "$HOME", etc.

ESC /

Goes to the ‘`/`’ directory (‘`ROOT-DIR`’).

ESC .

Goes to the ‘`..`’ directory (‘`UP-DIR`’).

ESC h

Goes to the ‘`~`’ ($HOME) directory (‘`HOME-DIR`’).

ESC i

Goes to the ‘`/usr/include`’ directory (‘`INCLUDE-DIR`’).

ESC 1

Goes to the ‘`/mnt/fd0`’ directory (‘`FIRST-FLOPPY-DIR`’).

ESC 2

Goes to the ‘`/mnt/fd1`’ directory (‘`SECOND-FLOPPY-DIR`’).

3.1.10 Compiling programs

F9, *^X m*

Runs the `make` command in the current directory.

^X b m

Runs the `make` command in background in the current directory.

See Section 3.9 [gitaction], page 21, for more information.

3.1.11 Sending/receiving ascii/binary mail

^C 2 a

Sends the current current ascii file by mail to an user supplied email address.

^C b 2 a

The same as above, the only difference being that the command runs in background.

^C 2 b

Sends the current current binary file by mail to an user supplied email address. The file is uuencoded first.

^C b 2 b

The same as above, the only difference being that the command runs in background.

ESC x r m

Runs the `emacs -f rmail` command. This will start the Emacs's '`rmail`' function so that you can read your mail.

3.1.12 Starting a sub-shell

^X z

Calls a sub-shell as specified by the *$GIT_SHELL* environment variable ('`SUB-SHELL`').

See Section 4.1 [Environment Variables], page 23, for more information.

3.1.13 Using grep and recursive grep

^X g

Searches using `grep` all the selected files for a given pattern ('`GREP`').

^X g

Searches recursively using `gitrgrep` all the user specified files and directories for a given pattern ('`RECURSIVE-GREP`').

See Section 3.11 [gitrgrep], page 22, for more information.

3.1.14 Locking your console

Having a lock feature might be a good idea and, since not all the UNIX systems provide one, `git` tries to get around the problem . . .

^X p

Prompts the user for a password and locks the console until the same password is reinserted ('`lock`').

3.1.15 Refreshing the screen contents

Sometimes your screen needs to be refreshed. Just think about what happens when somebody wants to talk with you and the talk daemon writes something like this

```
Message from Talk_Daemon@galei.cs.vu.nl at 12:15 ...
talk: connection requested by andrei@galei.cs.vu.nl.
talk: respond with:  talk andrei@galei.cs.vu.nl
```

on your screen. And sometimes you might also want to re-read the current directories. `git` provides some built-in commands for refreshing the screen contents.

^L

Re-read the directories contents re-displaying them using optimizations. Only those parts of the screen that have changed are repainted ('`refresh`').

^X l

Refresh the entire screen contents without using optimizations. Useful when some "nice" program wrote something on the screen, because `git` has no way to detect this ('`hard-refresh`').

3.1.16 Reseting your terminal

^X ^L

Calls `reset` in order to reset the terminal to its default settings (‘`TTY-RESET`’).

3.1.17 Mounting/unmounting file systems

People dealing with lots of files usually need to save/restore/copy files from/to other file systems. In order to be more efficient, `git` provides a set of key bindings for mounting and unmounting file systems. See Section 3.8 [gitmount], page 21, for more information.

The default key bindings set has been designed to work under `Linux`, but it can be easily changed for other UNIX systems with different device names. Reading the configuration file `.gitrc.common` should be enough. See Section 3.1.9.9 [Hot Keys], page 16, for more information.

As a convention, the ‘`/mnt`’ directory is used to store an empty subdirectory for each mountable file system. Each file system is actually mounted in its counterpart ‘`/mnt`’ subdirectory. Try to follow this convention since the `gitmount` script is heavily based on it. See Chapter 4 [Customization], page 23, for more information.

ESC m a

Calls `mount`(1) in order to mount the first floppy (‘`/dev/fd0`’) in the ‘`/mnt/fd0`’ directory (‘`MOUNT-A`’).

ESC m b

Calls `mount`(1) in order to mount the second floppy (‘`/dev/fd1`’) in the ‘`/mnt/fd1`’ directory (‘`MOUNT-B`’).

ESC m t

Calls `mount`(1) in order to mount the file system corresponding to the currently pointed to subdirectory. For example, if you are in the ‘`/mnt`’ directory and the cursor is on the ‘`fd0`’ subdirectory, the first floppy will be mounted (‘`MOUNT-THIS`’).

ESC u a

Calls `umount`(1) in order to unmount the first floppy (‘`/dev/fd0`’) (‘`UMOUNT-A`’).

ESC u b

Calls `umount`(1) in order to unmount the second floppy (‘`/dev/fd1`’) (‘`UMOUNT-B`’).

ESC u t

Calls `umount`(1) in order to unmount the file system mounted into the currently pointed to subdirectory (‘`UMOUNT-THIS`’). For example, if the current directory is ‘`/mnt`’ and the cursor points to the ‘`fd1`’ subdirectory, the second floppy will be unmounted.

3.1.18 Getting some useful system information

ESC S f

Calls `finger`(1) in order to display information about local and remote users (‘`FINGER`’).

ESC S m

Calls `mount`(1) in order to display a list of the currently mounted file systems (‘`MOUNTED-FILE-SYSTEMS`’).

ESC S q

Calls `quota`(1) in order to display a user file system disk quota and quota (‘`QUOTA`’).

ESC S s

Calls `df`(1) in order to get the status of the currently mounted file systems (‘`DISK-FREE-SPACE`’).

ESC S u

Calls `users`(1) in order to get the name of the currently logged in users (‘`USERS`’).

ESC S v

Calls `$GIT_VMSTAT`(1) in order to get the current virtual memory status. This is very system dependent, `Linux` uses `free`, other systems use `vmstat`, so the *$GIT_VMSTAT* variable is used to deal with this (‘`VIRTUAL-MEMORY-STATUS`’).

See Section 4.1 [Environment Variables], page 23, for more information.

ESC S w

Calls `who`(1) in order to find out who is on the system (‘`WHO`’).

3.1.19 How to look at the environment variables

^X E

Calls `env`(1) in order to display the current environment (‘`ENV`’).

3.1.20 Viewing/killing processes

There are at least two kinds of `ps`(1) utilities. One that accepts (more or less) combinations of the ’a’, ’u’, and ’x’ flags and another that accepts combinations of ’e’, ’f’ and ’l’ flags. Since is quite difficult to test which one works fine on a given `UNIX` system, `git` provides key bindings for both of them. Anyway, if your `ps`(1) fails to accept the predefined combinations, please take a look in its manual and then modify the `.gitrc.TERM` file as needed.

Since the number of possible combinations of flags in the `ps` command line is quite big and *very* system dependent, there is no real reason to display them all here. We are only interested in giving you a starting point in your search through the `.gitrc.TERM` file.

Note also that you can display a list of processes using `ps`(1) or browse through a list of them (killing as needed) using `gitps`. As a convention, we have used the same key sequence for a given set of `ps`(1) flags for both `ps`(1) and `gitps`, the only difference being that `ps`(1) keys end in an uppercase letter. See Section 3.2 [gitps], page 20, for more information.

Here there are the default key bindings for the 'e', 'f' and 'l' `ps`(1) flags combinations:

ESC P b, *ESC P c*, *ESC P e*

Calls `gitps` or `ps`(1) in order to browse through or display a list of currently running processes ('`GITPS`', '`PS`').

... and the default key bindings for the 'a', 'u' and 'x' `ps`(1) flags combinations:

ESC P a, *ESC P l*, *ESC P u ESC P x*, *ESC P y*

Calls `gitps` or `ps`(1) in order to browse through or display a list of currently running processes ('`GITPS`', '`PS`').

^X k

Calls `kill`(1) in order to kill a user specified process with a given signal ('`KILL`').

3.1.21 Synchronizing the file systems

^X S

Calls `sync`(1) in order to synchronize all the file systems ('`SYNC`').

3.1.22 Reading documentation

^X q

Reads a manual page. The user is prompted for its name ('`MAN`').

F1, ^X i

Reads an info documentation. The user is prompted for the documentation name ('`INFO`').

3.1.23 Exiting GNU Interactive Tools

F10, *^X ^C*, *^X c*

Exits GNU Interactive Tools ('`exit`').

3.2 The GIT process viewer/killer

`gitps` is an interactive process viewer/killer. It calls internally the `ps`(1) utility so that's the reason why `gitps` parameters are in fact `ps`(1) ones.

Running `gitps` it is self explanatory. Use the *arrows*, *PageUp*, *PageDown*, *Home*, *End*, *^N*, *^P*, *^V*, *ESC v* to move in the list, *^L* to refresh it and *F10* or *^G* to leave.

You can change these keys, just read the GITPS-Setup, GITPS-Color, GITPS-Monochrome and GITPS-Keys sections in the configuration files `.gitrc.TERM`.

3.3 The GIT ASCII/HEX file viewer

`gitview` is an ASCII/HEX file viewer. Use the *arrows*, *PageUp*, *PageDown*, *Home*, *End*, *^N*, *^P*, *^V*, *ESC v* to move in the file, *^L* to refresh the screen and *F10* or *^G* to leave.

You can change these keys, just read the GITVIEW-Setup, GITVIEW-Color, GITVIEW-Monochrome and GITVIEW-Keys sections in the configuration files `.gitrc.TERM`.

3.4 The GIT internal file compare utility

`gitcmp` is a file compare utility. It outputs to stderr, `git` catching its error messages this way.

`gtcmp` has been designed to be used in `$HOME/.gitrc.TERM` (`git` configuration file) and should not be used as a stand alone program.

3.5 The GIT key sequences display utility

`gitkeys` is a program that displays the key sequence sent by the pressed key. This is the key sequence received by `GIT` tools, so this program is useful when setting up the `.gitrc.TERM` configuration files.

3.6 The GIT file name match utility

`gitmatch` is a program that tries to match its first parameter (the file name) against some patterns, returning the number of the pattern that matched.

`gitmatch` is called by the `gitaction`/`.gitaction` scripts and was not designed to be used as a stand alone program.

3.7 The GIT wipe file utility

`gitwipe` is an utility for wiping files. It overwrites the file contents with a random sequence of numbers and then calls `sync`().

Note that `gitwipe` does *not* delete the file since (under `Linux` at least) the `sync`() system call might return before actually writing the new file contents to disk. Deleting the file might be dangerous because some file systems can detect that the blocks in the file are no longer used and never write them back to disk in order to improve performance. It is up to you to delete the file(s) at a later moment.

3.8 The GIT mount utility

`gitmount` is a script that allows you to mount any block device without specifying the file system type. You may now insert the floppy in the drive and type '`gitmount fd0'` and the first floppy will be mounted in the directory `/mnt/fd0`.

You don't need to know the file system type anymore. The directories `/mnt/fd0` and `/mnt/fd1` must exist. If you want to use `gitmount` with the block device `/dev/xxx` then the directory `/mnt/xxx` must exist too.

3.9 The GIT per file type action script

`gitaction` is a script that executes a different action for each file type specified. It is called by the `git` program when pressing *`F2`* or `^Xa`.

The first parameter is the current directory name and the second one is the file name to be matched against the default patterns. The matching is done using the gitmatch utility.

If you press *`F2`* or `^Xa` on a `*.c` file, `git` will compile it, if you press *`F2`* or `^Xa` on a `*.tar.gz` file, `git` will list the tar archive contents, if you press the same keys on a `*.gz` file, `git` will display its uncompressed contents on the screen, etc . . .

By default `gitaction` checks for the following patterns:
"*.cc" "*.c" "*.l" "*.y" "*.h" "*.s" "*.S" "*.o" "*.a" "*.sa" "Makefile" "makefile" "*.tar.gz" "*.tgz" "*.tar.z" "*.tar.Z" "*.taz" "*.tar" "*.gz" "*.z" "*.Z" "*.doc" "*.txt" "*.gif" "*.jpg" "*.tif" "*.bmp" "*.fli" "*.flc" compressed/uncompressed manual pages
and acts as appropriate. If no pattern is found, the file is displayed using `more`. Feel free to change this.

If you want to find out what the default action for each file type is (or if you want to modify it), just read/modify the `gitaction` script.

If you press *F2* or `^Xa` on a `*.gif` file or `*.jpg` file and you have the `zgv` utility installed, you will be able to see it. If you want to change the gif/jpeg viewer, all you need to do is to change its name in the `gitaction` script. I don't know a `*.bmp` or `*.tif` viewer. Feel free to add one in the `gitaction` script.

3.10 The GIT stdout to stderr redirection script

`gitredir` is a very small script that fools the `git` program, making it believe that the command started wrote something to standard error. It was not designed to be used as a stand alone program.

`gitredir` is useful for programs like `du`. We are interested in du's output and we would like to write it on the status line (especially the output of the 'du -s' command). Normally, this is not possible because 'du -s' will display the information on the standard output (`git` catches only standard error) and will exit with the exit code 0. `git` will display the standard error contents on the status bar only if the program exit status is not 0.

So, `gitredir` starts a program, sends its standard output through a pipe to a sub-shell which read from the pipe and write to standard error. After that, `gitredir` exits with the exit code 1. This way, `git` thinks that an error occured and display the standard error redirection file to the status bar. Stupid, but useful.

3.11 The GIT recursive grep script

`gitrgrep` is a very small script that calls `grep` recursively. It accepts `grep` like options / parameters, the only difference being that file specifications should be quoted:

```
gitrgrep main "*.c"
```

or

```
gitrgrep errno "*.c *.h"
```

4 Customizing GNU Interactive Tools

4.1 Environment Variables

The configuration files use shell environment variables to call the shell, editor, mail reader, compress and virtual memory status utility. That means that if you set *GIT_SHELL*, *GIT_EDITOR*, *GIT_RMAIL*, *GIT_COMPRESS* or *GIT_VMSTAT* to some value, that value will be used instead of the default one. The defaults are:

GIT_SHELL='/bin/sh'
GIT_EDITOR='vi'
GIT_RMAIL='emacs -f rmail'
GIT_PAGER='more'
GIT_COMPRESS='gzip -9'
GIT_VMSTAT='free'

If *SHELL* is defined, *GIT_SHELL* will be set to that value. If *EDITOR* is defined, *GIT_EDITOR* will be set to that value. If you want to change the default settings, put something like this into your `.profile`:

export GIT_SHELL='/usr/local/bin/bash'
export GIT_EDITOR='emacs'
export GIT_RMAIL='elm'
export GIT_PAGER='less'
export GIT_COMPRESS='compress'
export GIT_VMSTAT='vmstat'

4.2 Configuration Files

There is one configuration file per terminal type in `GIT`. The configuration file(s) reside in the user's home directory or (the default versions) in the directory `$(prefix)/lib` (usually `/usr/local/lib`).

Their generic name is `.gitrc.TERM`. `GIT` allows each terminal type to have its own configuration file (*TERM* is the value of the *TERM* environment variable (e.g '`vt102`'); for the `Linux` console the configuration file is `.gitrc.console`).

Since most of the key bindings are common to all the terminal types, a configuration file called `.gitrc.common` is parsed before parsing the normal `.gitrc.TERM` configuration file, the later one defining only those keys that are terminal specific. However, if a key binding is redefined in the `.gitrc.TERM` file, that binding will be used.

If the `GIT` package have been compiled without passing the '`--enable-terminfo`' option to the `configure` script and your system has a huge '`termcap`' database (`/etc/termcap`), you can copy the termcap definition(s) of your terminal(s) in a file called, lets say `.termcap` and put it in your home directory. After that, set your *TERMCAP* environment variable to point to it. You should add something like this to your `.profile`:

TERMCAP=`/home/mike/.termcap`

The interactive programs in the `GIT` package can run without such a file, but on systems with huge '`termcap`' databases, copying the definitions of the most used terminals in a local `.termcap` file will lead to a faster start.

The `.gitrc.TERM` it is first time searched in the home directory then, if not found, in the directory `$(prefix)/lib` (usually `/usr/local/lib`). The configuration file is structured on sections, each section containing variables in the following format:

'`variable-name`' = '`first-field`';'`second-field`'; ...

After the '`variable-name`' at least one space or tab is required. All characters after a '`#`' are ignored and if you comment a section name, the whole section is ignored.

Section names are enclosed in rectangular brackets ('`[`' and '`]`'). Note that this manual don't include them while refering to section names.

The `GIT` package contains three major programs: `git`, `gitps` and `gitview`. Each one has its own sections in the configuration files. There is also a global setup section called '`Setup`' that is used by all these programs.

4.2.1 Writing key sequences

`GIT` contains three interactive programs. Their names are: `git` (this is the file system browser), `gitps` (this is the process viewer/killer and `gitview` (this is the ASCII/HEX file viewer). Each one of these programs has its own set of key bindings.

The convention used in describing key bindings are very simple. Here there are some examples that will help you to understand them. The corresponding `Emacs` conventions will help you even more.

`^A` means keeping the Ctrl key down and pressing the `a` key (*C-a*).

The *ESC* character is represented as `^[` so that you can use the meta character (*M-*) where available (or the *ESC* key):

`^[a` corresponds to *M-a* (pressing the *ESC* key and then `a`).

The `^` character is represented as `^^`.

The *backspace* character is represented as `^_`.

The *Ctrl-SPACE* character (*C-SPC*) is represented as `^@`.

The space (*SPC*) character is represented as `^@`.

Note that the key bindings notation described here is only used in the configuration files. For the sake of readability this manual uses *ESC* for the *ESC* key, *SPC* for the *SPACE* key and *RET* for the *RETURN* (*ENTER*) key.

4.2.2 The global setup section

In this section the variables have only one field.

'`TempDirectory`'

This variable specifies the location of the temporary files created by `git`.

'`AnsiColorSequences`'

This variable should be set to '`ON`' if the terminal supports standard '`ANSI`' color sequences. Otherwise it should be '`OFF`'. If '`AnsiColorSequences`' is '`ON`', '`GITxxx-Color`' sections will be used in the configuration files `.gitrc.TERM`. Otherwise, `GIT` interactive programs will use the '`GITxxx-Monochrome`' sections.

'`UseLastScreenChar`'

This variable is used for terminals that can't write on the last character of the screen without scrolling the entire screen. If your terminal has no problem writing there (`Linux` console, vt100, vt102, xterm, ...) set it to '`ON`'. Otherwise (hpterm), it should be '`OFF`'.

'`ForegroundAtExit`' '`BackgroundAtExit`'

This variables are used to specify the foreground and background colors that `GIT` interactive programs should set at exit. It is useful when `GIT` is used under `X11` terminal emulators (xterm, color_xterm, etc) because these emulators usually work in reverse video and `GIT` can't figure out the foreground and background colors that shoud be restored at exit.

'`StartupScrollStep`'

This variable specifies the scroll step initial value for both panels.

4.2.3 git Sections

4.2.3.1 git Setup

In this section the variables have only one field.

'`StartupFileDisplayMode`'

This variable specifies the file specific information displayed at startup. It can be any of '`OwnerGroup`', '`DateTime`', '`Size`', '`Mode`' or '`FullName`'. Its value initially affects both panels but it can be changed separately afterward.

'`StartupFileSortMethod`'

This variable specifies the startup sort method. It can be any of '`Name`', '`Extension`', '`Size`', '`Date`', '`Mode`', '`OwnerId`', '`GroupId`', '`OwnerName`' or '`GroupName`'. Its value initially affects both panels but it can be changed separately afterward.

'`StartupLeftPanelPath`' '`StartupRightPanelPath`'

These variables specifies the startup path for each panel.

'`ConfirmOnExit`'

If this variable is '`ON`', the user is prompted for confirmation at exit.

'`HistoryFile`'

This variable specifies the history file name. The default value is `~/.githistory`.

'`InfoDisplay`'

If this variable is '`OFF`', auxiliary file informations are not displayed. This can be useful if you are using a very slow terminal.

'`FrameDisplay`'

If this variable is '`OFF`', the git frame is not displayed. This can be useful if you are using a very slow terminal.

'`LeadingDotMatch`'

If this variable is '`OFF`' when matching files for select-files-matching-pattern / unselect-files-matching-pattern then the leading '.' in the file name is matched only explicitly.

'`TypeSensitivity`'

If this variable is '`OFF`', colors are not used when displaying files. Normally, the information in the '`GIT-FTI`' section is used to display files with different colors, depending on their types. Note that '`TypeSensitivity`' is automatically set to '`OFF`' when

'AnsiColorSequences' is 'OFF'. See Section 4.2.4 [GIT-FTI], page 29, for mor information.

'NormalModeHelp' 'CommandLineModeHelp'

These variables describe the status bar contents for each git mode when no errors occurred. git can display on the status bar a help string and/or some system information (system type, hostname, machine type and the current date) using escape characters:

```
\s    ->    the system type
\h    ->    the host name
\m    ->    the machine type
\d    ->    the current date
```

See Section 3.1.2 [Modes], page 4, for more information.

4.2.3.2 Using git on colors displays

In this sections the variables have only one field.

These section allows you to customize the colors of git. Reading the .gitrc.TERM configuration file is self explanatory.

4.2.3.3 Using git on monochrome displays

In this sections the variables have only one field.

These section allows you to customize the appearance of git on monochrome displays. Reading the .gitrc.TERM configuration file is self explanatory.

4.2.3.4 Defining keys

These section describes the actions git takes when a specified key is pressed. A variable can have up to 6 fields separated by ';'. Each line in this section looks like:

'key-sequence' = 'command-name';'formatted-command';'new-dir';
'save-screen';'pause';'hide'

Note that you can't continue the variable fields description on the next line.

4.2.3.5 The key-sequence field

'key-sequence' is the key sequence associated with the given command. You can use any key sequence that doesn't start with an ascii character (0x20 to 0x7e).

Symbolic key names (*F0*, *F1*, *F2*, ... *F10*, *UP*, *DOWN*, *RIGHT*, *LEFT*, *INS*, *DEL*, *HOME*, *END*, *PGUP* and *PGDOWN*) can be used instead of the key sequence. If some keys don't have a 'termcap'/ 'terminfo' description (like the *F11*/*F12* keys on the Linux console) you can specify the key sequence in the usual way.

4.2.3.6 The command-name field

'command-name' is a command generic name. Even if it is not always used, the 'command-name' must be present (if a command is associated with a 'key-sequence'). If it is not, no action will be taken when pressing 'key-sequence'.

There are two types of commands in git: built-in commands and user defined commands. If the 'command-name' section contains a built-in command specification, the other fields are ignored.

Note that by convention built-in command names contain only lower case letters while user defined command names contain only upper case letters.

4.2.3.7 The formatted-command field

- '`formatted-command`' is a shell command which can contain some scanf like format specifiers. They are used to get the current entry name, owner, group, mode, etc.

Note that using uppercase '`format specifiers`' you will be able to access the other panel path, file and directory names, etc.

These are the available '`format specifiers`':

4.2.3.8 The %s format specifier

The format of %s is: %s{question,default_answer}.

When `git` encounters a %s in the '`formatted-command`' it asks the user the question '`question`' whose default answer is '`default_answer`' and replaces the '`%s{ , }`' with the user's answer. Both '`question`' and '`default_answer`' can contain any other '`format specifiers`' except %s.

Note that there should be no spaces between %s and '{'.

4.2.3.9 The %f format specifier

`git` will replace %f with the current directory entry name only if it is a file (not a directory).

4.2.3.10 The %d format specifier

`git` will replace %d with the current directory entry name only if it is a directory (not a file).

4.2.3.11 The %l format specifier

`git` will replace %l with the current directory entry name only if it is a symbolic link with no target.

4.2.3.12 The %t format specifier

`git` will replace %t with the current directory entry name only if it is a named pipe.

4.2.3.13 The %z format specifier

`git` will replace %z with the current directory entry name only if it is a socket.

4.2.3.14 The %a format specifier

`git` will always replace %a with the current directory entry name.

4.2.3.15 The %m format specifier

`git` will always replace %m with the current file mode.

4.2.3.16 The %g format specifier

`git` will always replace %g with the current file group.

4.2.3.17 The %o format specifier

`git` will always replace %o with the current file owner.

4.2.3.18 The %p format specifier

`git` will always replace %p with the current panel path.

4.2.3.19 The %b format specifier

`git` will always replace %b with the current panel directory name.

4.2.3.20 The %i format specifier

`git` will always replace %i with all the current panel selected file names.

4.2.3.21 The %? format specifier

The format of %? is: %?{confirmation}.

`git` uses this format specifier only to ask for confirmation before expanding / executing the current command. The '`confirmation`' string is displayed and, if the user doesn't confirm, the command is aborted. Otherwise, %?{confirmation} expands to a null string and the command is expanded / executed normally.

4.2.3.22 The new-dir field

If the '`formatted-command`' successfully exits (exit code = 0) or it has no body and this field is present then '`new-dir`' will become the current panel directory.

The character '`~`' used at the beginning of the '`new-dir`' field is replaced by the user's home directory.

4.2.3.23 The save-screen field

This field is a character (usually 'y' or 'n') that tells `git` to save ('y') or not to save ('n') the terminal's screen after executing the '`formatted-command`'. Saving the screen is not necessary while editing or viewing a file because the information left after the editor or the viewer exits is not important. Saving the screen means that that screen will be restored before the execution of the next command. Currently this field is used only if you are working as a super user under `Linux` on a virtual console. Its default value is 'y'.

4.2.3.24 The pause field

Users may wish to read some commands's results before repainting the panels. If this field is present git will wait for a key to be pressed before restoring the panels. Its default value is 'n'.

4.2.3.25 The hide field

Some commands that don't displaying any useful information if successfully complete their execution: `mount`, `chmod`, `chown`, `chgrp`, `sync` ... and, if an error occurs, a line or two are sent to stderr. If this option is 'y', the stdout and stderr will be redirected to some files (stdout.pid and stderr.pid, where pid is `git`'s pid) and only if the command's exit code is not 0, the stderr.pid file will be displayed, line by line, onto the status bar. This way

the panels will not be deleted and then repainted and the command appears to be built-in. stdout.pid and stderr.pid are created in the '`TempDirectory`' specified in the '`Setup`' section. Its default value is 'n'.

4.2.4 Setting up colors for different file types

This sections contains entries of the form:

```
'pattern' = 'foreground'; 'background'; 'brightness'
```

where '`pattern`' is a file name matching pattern, '`foreground`', '`background`' and '`brightness`' are the color specification to be used when a file whose name match the given '`pattern`' is displayed in a panel. Colors can be turned off using the '`TypeSensitivity`' variable in the '`GIT-Setup`' seection.

4.2.5 gitps Sections

4.2.5.1 gitps Setup

In this section the variables have only one field.

4.2.5.2 Using gitps on color displays

In this sections the variables have only one field.

These section allows you to customize the colors of `gitps`. Reading the `.gitrc.TERM` configuration file is self explanatory.

4.2.5.3 Using gitps on monochrome displays

In this sections the variables have only one field.

These section allows you to customize the appearance of `gitps` on monochrome displays. Reading the `.gitrc.TERM` configuration file is self explanatory.

4.2.5.4 Defining keys

4.2.6 gitview Sections

4.2.6.1 gitview Setup

In this section the variables have only one field.

4.2.6.2 Using gitview on color displays

In this sections the variables have only one field.

These section allows you to customize the colors of `gitview`. Reading the `.gitrc.TERM` configuration file is self explanatory.

4.2.6.3 Using gitview on monochrome displays

In this sections the variables have only one field.

These section allows you to customize the appearance of `gitview` on monochrome displays. Reading the `.gitrc.TERM` configuration file is self explanatory.

4.2.6.4 Defining keys

5 GNU Interactive Tools limitations

Background execution is not fully supported. That means you may specify an & terminated command line in the configuration file but the result of the background executed command (stdout and stderr redirection), will be overwritten by the result of newer commands and, if an error occurs, you will not see it.

When `git` is compiled for `Linux`, the default built-in color descriptions are for color monitors, so you can't (decently) run `git` on a b/w monitor without the `.gitrc.TERM` file correctly configured. `.gitrc.TERM` should be configured with '`AnsiColorSequences`' = OFF.

Job support is implemented only in `git`.

Due to the fact that the ';' character is used as a field separator in the configuration files, you can't write something like that in the `.gitrc.TERM` files:

^AAA = SHOW-USERS-AND-GROUPS; more /etc/passwd; more /etc/group

because 'more /etc/group' will be considered as a directory to switch to. You must write a small script instead:

#!/bin/sh

more /etc/passwd more /etc/group

Supposing the script name is `show_ug`, the `.gitrc.TERM` line will look like this:

^AAA = SHOW-USERS-AND-GROUPS; show_ug

6 GNU Interactive Tools bugs

Any questions, comments, or bug reports, should be emailed to the authors. Please be sure to include the version number. The email addresses are:

tudor@chang.pub.ro
pink@pub.ro

Key Index

^

D

E

F

H

I

P

T

U

Command Index

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Variable Index

Concept Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Short Contents

Table of Contents