EXT2ED - The Extended-2 filesystem editor - User's guide

Gadi Oxman, tgud@tochnapc2.technion.ac.il
v0.1, August 3 1995

This is only the initial version of this document. It may be unclear at some places. Please send me feedback with anything regarding to it.

Contents

1 About EXT2ED documentation

The EXT2ED documentation consists of three parts:

- The ext2 filesystem overview.
- The EXT2ED user's guide.
- The EXT2ED design and implementation.

If you intend to used EXT2ED, I strongly suggest that you would be familiar with the material presented in the `ext2 filesystem overview` as well.

If you also intend to browse and modify the source code, I suggest that you will also read the article `The EXT2ED design and implementation`, as it provides a general overview of the structure of my source code.

2 Introduction

EXT2ED is a "disk editor" for the ext2 filesystem. Its purpose is to show you the internal structures of the ext2 filesystem in an rather intuitive and logical way, so that it will be easier to "travel" between the various internal filesystem structures.

3 Basic concepts in EXT2ED

Two basic concepts in EXT2ED are `commands` and `types`.

EXT2ED is object-oriented in the sense that it defines objects in the filesystem, like a `super-block` or a `directory`. An object is something which "knows" how to handle some aspect of the filesystem.

Your interaction with EXT2ED is done through `commands` which EXT2ED accepts. There are three levels of commands:

- General Commands
- Extended-2 Filesystem general commands
- Type specific commands

The General commands are always available.

The ext2 general commands are available only when editing an ext2 filesystem.

The Type specific commands are available when editing a specific object in the filesystem. Each object typically comes with its own set of internal variables, and its own set of commands, which are fine tuned handle the corresponding structure in the filesystem.

4 Running EXT2ED

Running EXT2ED is as simple as typing `ext2ed` from the shell prompt. There are no command line switches.

When first run, EXT2ED parses its configuration file, `ext2ed.conf`. This file must exist.

When the configuration file processing is done, EXT2ED screen should appear on the screen, with the command prompt `ext2ed>` displayed.

5 EXT2ED user interface

EXT2ED uses the *ncurses* library for screen management. Your screen will be divided into four parts, from top to bottom:

- Title window
- Status window
- Main editing window
- Command window

The title window just displays the current version of EXT2ED.

The status window will display various information regarding the state of the editing at this point.

The main editing window is the place at which the actual data will be shown. Almost every command will cause some display at this window. This window, as opposed to the three others, is of variable length - You always look at one page of it. The current page and the total numbers of pages at this moment is displayed at the status window. Moving between pages is done by the use of the `pgdn` and `pgup` commands.

The command window is at the bottom of the screen. It always displays a command prompt `ext2ed>` and allows you to type a command. Feedback about the commands entered is displayed to this window also.

EXT2ED uses the *readline* library while processing a command line. All the usual editing keys are available. Each entered command is placed into a history of commands, and can be recalled later. Command Completion is also supported - Just start to type a command, and press the completion key.

Pressing `enter` at the command window, without entering a command, recalls the last command. This is useful when moving between close entries, in the `next` command, for example.

6 Getting started

6.1 A few precautions

EXT2ED is a tool for filesystem `editing`. As such, it can be `dangerous`. The summary to the subsections below is that `You must know what you are doing`.

6.1.1 A mounted filesystem

EXT2ED is not designed to work on a mounted filesystem - It is complicated enough as it is; I didn't even try to think of handling the various race conditions. As such, please respect the following advice:

`Do not use EXT2ED on a mounted filesystem !`

EXT2ED will not allow write access to a mounted filesystem. Although it is fairly easy to change EXT2ED so that it will be allowed, I hereby request again- EXT2ED is not designed for that action, and will most likely corrupt data if used that way. Please don't do that.

Concerning read access, I chose to leave the decision for the user through the configuration file option `AllowMountedRead`. Although read access on a mounted partition will not do any damage to the filesystem, the data displayed to you will not be reliable, and showing you incorrect information may be as bad as corrupting the filesystem. However, you may still wish to do that.

6.1.2 Write access

Considering the obvious sensitivity of the subject, I took the following actions:

1. EXT2ED will always start with a read-only access. Write access mode needs to be specifically entered by the `enablewrite` command. Until this is done, no write will be allowed. Write access can be disabled at any time with `disablewrite`. When `enablewrite` is issued, the device is reopened in read-write mode. Needless to say, the device permissions should allow that.
2. As a second level of protection, you can disallow write access in the configuration file by using the `AllowChanges off` configuration option. In this case, the `enablewrite` command will be refused.
3. When write access is enabled, the data will never change immediately. Rather, a specific `writedata` command is needed to update the object in the disk with the changed object in memory.
4. In addition, A logging option is provided through the configuration file options `LogChanges` and `LogFile`. With logging enabled, each change to the disk will be logged at a very primitive level - A hex dump of the original data and of the new written data. The log file will be a text file which is easily readable, and you can make use of it to undo any changes which you made (EXT2ED doesn't make use of the log file for that purpose, it just logs the changes).

Please remember that this is only the initial release of EXT2ED, and it is not very much tested - It is reasonable to assume that `there are bugs`. However, the logging option above can offer protection even from this unfortunate case. Therefor, I highly recommend that at least when first working with EXT2ED, the logging option will be enabled, despite the disk space which it consumes.

6.2 The help command

When loaded, EXT2ED will show a short help screen. This help screen can always be retrieved by the command `help`. The help screen displays a list of all the commands which are available at this point. At startup, only the `General commands` are available. This will change with time, since each object has its own commands. Thus, commands which are available now may not be available later. Using `help` *command* will display additional information about the specific command *command*.

6.3 The setdevice command

The first command that is usually entered to EXT2ED is the `setdevice` command. This command simply tells EXT2ED on which device the filesystem is present. For example, suppose my ext2 filesystem is on the first partition of my ide disk. The command will be:

```
setdevice /dev/hda1
```

The following actions will take place in the following order:

1. EXT2ED will check if the partition is mounted. If the partition is mounted (`highly not recommended`), the accept/reject behavior will be decided by the configuration file. Cross reference section 6.1.1.
2. The specified device will be opened in read-only mode. The permissions of the device should be set in a way that allows you to open the device for read access.
3. Autodetection of an ext2 filesystem will be made by searching for the ext2 magic number in the main superblock.
4. In the case of a successful recognition of an ext2 filesystem, the ext2 filesystem specific commands and the ext2 specific object definitions will be registered. The object definitions will be read at run time from a file specified by the configuration file.

   In case of a corrupted ext2 filesystem, it is quite possible that the main superblock is damaged and autodetection will fail. In that case, use the configuration option `ForceExt2 on`. This is not the default case since EXT2ED can be used at a lower level to edit a non-ext2 filesystem.
5. In a case of a successful autodetection, essential information about the filesystem such as the block size will be read from the superblock, unless the used overrides this behavior with an configuration option (not recommended). In that case, the parameters will be read from the configuration file.

   In a case of an autodetection failure, the essential parameters will be read from the configuration file.

Assuming that you are editing an ext2 filesystem and that everything goes well, you will notice that additional commands are now available in the help screen, under the section `ext2 filesystem general commands`. In addition, EXT2ED now recognizes a few objects which are essential to the editing of an ext2 filesystem.

7 Two levels of usage

7.1 Low level usage

This section explains what EXT2ED provides even when not editing an ext2 filesystem.

Even at this level, EXT2ED is more than just a hex editor. It still allows definition of objects and variables in run time through a user file, although of-course the objects will not have special fine tuned functions connected to them. EXT2ED will allow you to move in the filesystem using `setoffset`, and to apply an object definition on a specific place using `settype` *type*. From this point and on, the object will be shown `in its native form` - You will see a list of the variables rather than just a hex dump, and you will be able to change each variable in the intuitive form `set variable=value`.

To define objects, use the configuration option `AlternateDescriptors`.

There are now two forms of editing:

- Editing without a type. In this case, the disk block will be shown as a text+hex dump, and you will be able to move along and change it.
- Editing with a type. In this case, the object's variables will be shown, and you will be able to change each variable in its native form.

7.2 High level usage

EXT2ED was designed for the editing of the ext2 filesystem. As such, it "understands" the filesystem structure to some extent. Each object now has special fine tuned 'C' functions connected to it, which knows how to display it in an intuitive form, and how the object fits in the general design of the ext2 filesystem. It is of-course much easier to use this type of editing. For example:

> `Issue` *group 2* `to look at the main copy of the third group block descriptor. With` *gocopy 1* `you can move to its first backup copy, and with` *inode* `you can start editing the inode table of the above group block. From here, if the inode corresponds to a file, you can use` *file* `to edit the file in a "continuous" way, using` *nextblock* `to pass to its next block, letting EXT2ED following by itself the direct blocks, indirect blocks, ..., while still preserving the actual view of the exact block usage of the file.`

The point is that the "tour" of the filesystem will now be synchronic rather than asynchronic - Each object has the "links" to pass between connected logical structures, and special fine-tuned functions to deal with it.

8 General commands

I will now start with a systematic explanation of the general commands. Please feel free to experiment, but take care when using the `enablewrite` command.

Whenever a command syntax is specified, arguments which are optional are enclosed with square brackets.

Please note that in EXT2ED, each command can be overridden by a specific object to provide special fine-tuned functionality. In general, I was attempting to preserve the similarity between those functions, which are accessible by the same name.

8.1 disablewrite

```
Syntax: disablewrite
```

`disablewrite` is used to reopen the device with read-only access. When first running EXT2ED, the device is opened in read-only mode, and an explicit `enablewrite` is required for write access. When finishing with changing, a `disablewrite` is recommended for safety. Cross reference section 8.1.

8.2 enablewrite

```
Syntax: enablewrite
```

`enablewrite` is used to reopen the device with read-write access. When first running EXT2ED, the device is opened in read-only mode, and an explicit `enablewrite` is required for write access. `enablewrite` will fail if write access is disabled from the configuration file by the `AllowChanges off` configuration option. Even after `enablewrite`, an explicit `writedata` is required to actually write the new data to the disk. When finishing with changing, a `disablewrite` is recommended for safety. Cross reference section 8.2.

8.3 help

```
Syntax: help [command]
```

The `help` command is described at section 6.2.

8.4 next

```
Syntax: next [number]
```

This section describes the *general command* `next`. `next` is overridden by several types in EXT2ED, to provide fine-tuned functionality.

The `next general command` behavior is depended on whether you are editing a specific object, or none.

- In the case where Type is `none` (The current type is showed on the status window by the `show` command), `next` passes to the next *number* bytes in the current edited block. If *number* is not specified, *number=1* is assumed.
- In the case where Type is defined, the `next` commands assumes that you are editing an array of objects of that type, and the `next` command will just pass to the next entry in the array. If *number* is defined, it will pass *number* entries ahead.

8.5 pgdn

    Syntax: pgdn

Usually the edited data doesn't fit into the visible main window. In this case, the status window will indicate that there is more to see "below" by the message `Page x of y`. This means that there are y pages total, and you are currently viewing the x page. With the `pgdn` command, you can pass to the next available page.

8.6 pgup

    Syntax: pgup

`pgup` is the opposite of `pgdn` - It will pass to the previous page. Cross reference section 8.5.

8.7 prev

    Syntax: prev [number]

`prev` is the opposite of `next`. Cross reference section 8.4.

8.8 recall

    Syntax: recall object

`recall` is the opposite of `remember`. It will place you at the place you where when saving the object position and type information. Cross reference section 8.10.

8.9 redraw

    Syntax: redraw

Sometimes the screen display gets corrupted. I still have problems with this. The `redraw` command simply redraws the entire display screen.

8.10 remember

    Syntax: remember object

EXT2ED provides you `memory` of objects; While editing, you may reach an object which you will like to return to later. The `remember` command will store in memory the current place and type of the object. You can return to the object by using the `recall` command. Cross reference section 8.8.

`Note:`

- When remembering a `file` or a `directory`, the corresponding inode will be saved in memory. The basic reason is that the inode is essential for finding the blocks of the file or the directory.

8.11 set

```
Syntax: set [text || hex] arg1 [arg2 arg3 ...]

or

Syntax: set variable=value
```

The `set` command is used to modify the current data. The `set general command` behavior is depended on whether you are editing a specific object, or none.

- In the case where Type is `none`, the first syntax should be used. The set command affects the data starting at the current highlighted position in the edited block.
  - When using the `set hex` command, a list of hexadecimal bytes should follow.
  - When using the `set text` command, it should be followed by a text string.

  Examples:

  ```
  set hex 09 0a 0b 0c 0d 0e 0f
  set text Linux is just great !
  ```

- In the case where Type is defined, the second syntax should be used. The set commands just sets the variable *variable* with the value *value*.

In any case, the data is only changed in memory. For an actual update to the disk, use the `writedata` command.

8.12 setdevice

```
Syntax: setdevice device
```

The `setdevice` command is described at section 6.3.

8.13 setoffset

```
Syntax: setoffset [block || type] [+|-]offset
```

The `setoffset` command is used to move asynchronically inside the file system. It is considered a low level command, and usually should not be used when editing an ext2 filesystem, simply because movement is better utilized through the specific ext2 commands.

The `offset` is in bytes, and meanwhile should be positive and smaller than 2GB.

Use of the `block` modifier changes the counting unit to block.

Use of the `+ or -` modifiers signals that the offset is relative to the current position.

use of the `type` modifier is allowed only with relative offset. This modifier will multiply the offset by the size of the current type.

8.14 settype

```
Syntax: settype type || [none | hex]
```

The `settype` command is used to move apply the object definitions of the type *type* on the current position. It is considered a low level command and usually should not be used when editing an ext2 filesystem since EXT2ED provides better tools. It is of-course very useful when editing a non-ext2 filesystem and using user-defined objects.

When *type* is *hex* or *none*, the data will be displayed as a hex and text dump.

8.15 show

```
Syntax: show
```

The `show` command will show the data of the current object at the current position on the main display window. It will also update the status window with type specific information. It may be necessary to use `pgdn` and `pgup` to view the entire data.

8.16 writedata

```
Syntax: writedata
```

The `writedata` command will update the disk with the object data that is currently in memory. This is the point at which actual change is made to the filesystem. Without this command, the edited data will not have any effect. Write access should be allowed for a successful update.

9 Editing an ext2 filesystem

In order to edit an ext2 filesystem, you should, of course, know the structure of the ext2 filesystem. If you feel that you lack some knowledge in this area, I suggest that you do some of the following:

- Read the supplied ext2 technical information. I tried to summarize the basic information which is needed to get you started.
- Get the slides that Remy Card (The author of the ext2 filesystem) prepared concerning the ext2 filesystem.
- Read the kernel sources.

At this point, you should be familiar with the following terms: `block, inode, superblock, block groups, block allocation bitmap, inode allocation bitmap, group descriptors, file, directory.`Most of the above are objects in EXT2ED.

When editing an ext2 filesystem it is recommended that you use the ext2 specific commands, rather then the general commands `setoffset` and `settype`, mainly because:

1. In most cases it will be unreliable, and will display incorrect information.

   Sometimes in order to edit an object, EXT2ED needs the information of some other related objects. For example, when editing a directory, EXT2ED needs access to the inode of the edited directory. Simply setting the type to a directory `will be unreliable`, since the object assumes that you passed through its inode to reach it, and expects this information, which isn't initialized if you directly set the type to a directory.

2. EXT2ED offers far better tools for handling the ext2 filesystem using the ext2 specific commands.

10 ext2 general commands

The `ext2 general commands` are available only when you are editing an ext2 filesystem. They are `general` in the sense that they are not specific to some object, and can be invoked anytime.

10.1 super

```
Syntax: super
```

The `super` command will "bring you" to the main superblock copy. It will automatically set the object type to `ext2_super_block`. Then you will be able to view and edit the superblock. When you are in the superblock, other commands will be available.

10.2 group

```
Syntax: group [number]
```

The `group` command will "bring you" to the main copy of the *number* group descriptor. It will automatically set the object type to `ext2_group_desc`. Then you will be able to view and edit the group descriptor entry. When you are there, other commands will be available.

10.3 cd

```
Syntax: cd path
```

The `cd` command will let you travel in the filesystem in the nice way that the mounted filesystem would have let you.

The `cd` command is a complicated command. Although it may sound simple at first, an implementation of a typical cd requires passing through the group descriptors, inodes, directory entries, etc. For example:

The innocent cd /usr command can be done by using more primitive EXT2ED commands in the following way (It is implemented exactly this way):

1. Using `group 0` to go to the first group descriptor.

2. Using `inode` to get to the Bad blocks inode.
3. Using `next` to pass to the root directory inode.
4. Using `dir` to see the directory.
5. Using `next` until we find the directory usr.
6. Using `followinode` to pass to the inode corresponding to usr.
7. Using `dir` to see the directory of /usr.

And those commands aren't that primitive; For example, the tracing of the blocks which belong to the root directory is done automatically by the dir command behind the scenes, and the followinode command will automatically "run" to the correct group descriptor in order to find the required inode.

The path to the `general cd` command needs to be a full pathname - Starting from `/`. The `cd` command stops at the last reachable point, which can be a directory entry, in which case the type will be set to `dir`, or an inode, in which case the type will be set to `ext2_inode`. Symbolic links (Only fast symbolic links, meanwhile) are automatically followed (if they are not across filesystems, of-course). If the type is set to `dir`, you can use a path relative to the "current directory".

11 The superblock

The superblock can always be reached by the ext2 general command `super`. Cross reference section 10.1.

The status window will show you which copy of the superblock copies you are currently editing.

The main data window will show you the values of the various superblock variables, along with some interpretation of the values.

Data can be changed with the `set` and `writedata` commands.

```
For example, set s_r_blocks_count=1400 will reserve 1400 blocks for root.
```

11.1 gocopy

```
Syntax: gocopy number
```

The `gocopy` command will "bring you" to the backup copy *number* of the superblock copies. `gocopy 0`, for example, will bring you to the main copy.

11.2 setactivecopy

```
Syntax: setactivecopy
```

The `setactivecopy` command will copy the contents of the current superblock copy onto the contents of the main copy. It will also switch to editing of the main copy. No actual data is written to disk, of-course, until you issue the `writedata` command.

12 The group descriptors

The group descriptors can be edited by the `group` command.

The status window will indicate the current group descriptor, the total number of group descriptors (and hence of group blocks), and the backup copy number.

The main data window will just show you the values of the various variables.

Basically, you can use the `next` and `prev` commands, along with the `set` command, to modify the group descriptors.

The group descriptors object is a junction, from which you can reach:

- The inode table of the corresponding block group (the `inode` command)
- The block allocation bitmap (the `blockbitmap` command)
- The inode allocation bitmap (the `inodebitmap` command)

12.1 blockbitmap

```
Syntax: blockbitmap
```

The `blockbitmap` command will let you edit the block bitmap allocation block of the current group block.

12.2 entry

```
Syntax: entry number
```

The `entry` command will move you to the *number* group descriptor in the group descriptors table.

12.3 inode

```
Syntax: inode
```

The `inode` command will pass you to the first inode in the current group block.

12.4 inodebitmap

```
Syntax: inodebitmap
```

The `inodebitmap` command will let you edit the inode bitmap allocation block of the current group block.

12.5 next

```
Syntax: next [number]
```

The **next** command will pass to the next *number* group descriptor. If *number* is omitted, *number=1* is assumed.

12.6 prev

```
Syntax: prev [number]
```

The **prev** command will pass to the previous *number* group descriptor. If *number* is omitted, *number=1* is assumed.

12.7 setactivecopy

```
Syntax: setactivecopy
```

The **setactivecopy** command copies the contents of the current group descriptor, to its main copy. The updated main copy will then be shown. No actual change is made to the disk until you issue the **writedata** command.

13 The inode

An inode can be reached by the following two ways:

- Using **inode** from the corresponding group descriptor.
- Using **followinode** from a directory entry.
- Using the **cd** command with the pathname to the file.
  For example, **cd /usr/src/ext2ed/ext2ed.h**

The status window will indicate:

- The current global inode number.
- The total total number of inodes.
- On which block group the inode is allocated.
- The total number of inodes in this group block.
- The index of the current inode in the current group block.
- The type of the inode (file, directory, special, etc).

The main data window, in addition to the list of variables, will contain some interpretations on the right side.

If the inode corresponds to a file, you can use the `file` command to edit the file.

If the inode is an inode of a directory, you can use the `dir` command to edit the directory.

13.1 dir

```
Syntax: dir
```

If the inode mode corresponds to a directory (shown on the status window), you can enter directory mode editing by using `dir`.

13.2 entry

```
Syntax: entry number
```

The `entry` command will move you to the *number* inode in the current inode table.

13.3 file

```
Syntax: file
```

If the inode mode corresponds to a file (shown on the status window), you can enter file mode editing by using `file`.

13.4 group

```
Syntax: group
```

The `group` command is used to go to the group descriptor of the current group block.

13.5 next

```
Syntax: next [number]
```

The `next` command will pass to the next *number* inode. If *number* is omitted, *number=1* is assumed.

13.6 prev

```
Syntax: prev [number]
```

The `prev` command will pass to the previous *number* inode. If *number* is omitted, *number=1* is assumed.

14 The file

When editing a file, EXT2ED offers you a both a continuous and a true fragmented view of the file - The file is still shown block by block with the true block number at each stage and EXT2ED offers you commands which allow you to move between the `file blocks`, while finding the allocated blocks by using the inode information behind the scenes.

Aside from this, the editing is just a `hex editing` - You move the cursor in the current block of the file by using `next` and `prev`, move between blocks by `nextblock` and `prevblock`, and make changes by the `set` command. Note that the set command is overridden here - There are no variables. The `writedata` command will update the current block to the disk.

Reaching a file can be done by using the `file` command from its inode. The `inode` can be reached by any other means, for example, by the `cd` command, if you know the file name.

The status window will indicate:

- The global block number.
- The internal file block number.
- The file offset.
- The file size.
- The file inode number.
- The indirection level - Whether it is a direct block (0), indirect (1), etc.

The main data window will display the file either in hex mode or in text mode, select-able by the `display` command.

In hex mode, EXT2ED will display offsets in the current block, along with a text and hex dump of the current block.

In either case the `current place` will be highlighted. In the hex mode it will be always highlighted, while in the text mode it will be highlighted if the character is display-able.

14.1 block

```
Syntax: block block_num
```

The `block` command is used to move inside the file. The *block_num* argument is the requested internal file block number. A value of 0 will reach the beginning of the file.

14.2 display

```
Syntax: display [text || hex]
```

The `display` command changes the display mode of the file. `display hex` will switch to `hex mode`, while `display text` will switch to text mode. The default mode when no `display` command is issued is `hex mode`.

14.3 inode

```
Syntax: inode
```

The `inode` command will return to the inode of the current file.

14.4 next

```
Syntax: next [num]
```

The `next` command will pass to the next byte in the file. If *num* is supplied, it will pass to the next *num* bytes.

14.5 nextblock

```
Syntax: nextblock [num]
```

The `nextblock` command will pass to the next block in the file. If *num* is supplied, it will pass to the next *num* blocks.

14.6 prev

```
Syntax: prev [num]
```

The `prev` command will pass to the previous byte in the file. If *num* is supplied, it will pass to the previous *num* bytes.

14.7 prevblock

```
Syntax: prevblock [num]
```

The `nextblock` command will pass to the previous block in the file. If *num* is supplied, it will pass to the previous *num* blocks.

14.8 offset

```
Syntax: offset file_offset
```

The `offset` command will move to the specified offset in the file.

14.9 set

```
Syntax: set [text || hex] arg1 [arg2 arg3 ...]
```

The `file set` command is working like the `general set command`, with `type=none`. There are no variables.

14.10 writedata

```
Syntax: writedata
```

The `writedata` command will update the current file block in the disk.

15 The directory

When editing a file, EXT2ED analyzes for you both the allocation blocks of the directory entries, and the directory entries.

Each directory entry is displayed on one row. You can move the highlighted entry with the usual `next` and `prev` commands, and "dive in" with the `followinode` command.

The status window will indicate:

- The directory entry number.
- The total number of directory entries in this directory.
- The current global block number.
- The current offset in the entire directory - When viewing the directory as a continuous file.
- The inode number of the directory itself.
- The indirection level - Whether it is a direct block (0), indirect (1), etc.

15.1 cd

```
Syntax: cd [path]
```

The `cd` command is used in the usual meaning, like the global cd command.

- If *path* is not specified, the current directory entry is followed.
- *path* can be relative to the current directory.
- *path* can also end up in a file, in which case the file inode will be reached.
- Symbolic link (fast only, meanwhile) is automatically followed.

15.2 entry

```
Syntax: entry [entry_num]
```

The `entry` command sets *entry_num* as the current directory entry.

15.3 followinode

```
Syntax: followinode
```

The `followinode` command will move you to the inode pointed by the current directory entry.

15.4 inode

```
Syntax: inode
```

The `inode` command will return you to the parent inode of the whole directory listing.

15.5 next

```
Syntax: next [num]
```

The `next` command will pass to the next directory entry. If *num* is supplied, it will pass to the next *num* entries.

15.6 prev

```
Syntax: prev [num]
```

The `prev` command will pass to the previous directory entry. If *num* is supplied, it will pass to the previous *num* entries.

15.7 writedata

```
Syntax: writedata
```

The `writedata` command will write the current directory entry to the disk.

16 The block allocation bitmap

The `block allocation bitmap` of any block group can be reached from the corresponding group descriptor.

You will be offered a bit listing of the entire blocks in the group. The current block will be highlighted and its number will be displayed in the status window.

A value of "1" means that the block is allocated, while a value of "0" signals that it is free. The value is also interpreted in the status window. You can use the usual `next/prev` commands, along with the `allocate/deallocate` commands.

16.1 allocate

```
Syntax: allocate [num]
```

The `allocate` command allocates *num* blocks, starting from the highlighted position. If *num* is not specified, *num=1* is assumed. Of-course, no actual change is made until you issue a `writedata` command.

16.2 deallocate

```
Syntax: deallocate [num]
```

The `deallocate` command deallocates *num* blocks, starting from the highlighted position. If *num* is not specified, *num=1* is assumed. Of-course, no actual change is made until you issue a `writedata` command. `writedata` command.

16.3 entry

```
Syntax: entry [entry_num]
```

The `entry` command sets the current highlighted block to *entry_num*.

16.4 next

```
Syntax: next [num]
```

The `next` command will pass to the next bit, which corresponds to the next block. If *num* is supplied, it will pass to the next *num* bits.

16.5 prev

```
Syntax: prev [num]
```

The `prev` command will pass to the previous bit, which corresponds to the previous block. If *num* is supplied, it will pass to the previous *num* bits.

17 The inode allocation bitmap

The `inode allocation bitmap` is very similar to the block allocation bitmap explained above. It is also reached from the corresponding group descriptor. Please refer to section 16.

18 Filesystem size limitation

While an ext2 filesystem has a size limit of `4 TB`, EXT2ED currently `can't` handle filesystems which are `bigger than 2 GB`.

I am sorry for the inconvenience. This will hopefully be fixed in future releases.

19 Copyright

The entire EXT2ED project is based, of-course, on the kernel sources. The `ext2.descriptors` distributed with EXT2ED is a slightly modified version of the main ext2 include file, /usr/include/linux/ext2_fs.h. Follows the original copyright:

```
/*
 *  linux/include/linux/ext2_fs.h
 *
 * Copyright (C) 1992, 1993, 1994, 1995
 * Remy Card (card@masi.ibp.fr)
 * Laboratoire MASI - Institut Blaise Pascal
 * Universite Pierre et Marie Curie (Paris VI)
 *
 *  from
 *
 *  linux/include/linux/minix_fs.h
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 */
```

20 Acknowledgments

EXT2ED was constructed as a student project in the software laboratory of the faculty of electrical-engineering in the `Technion - Israel's institute of technology`.

At first, I would like to thank `Avner Lottem` and `Doctor Ilana David` for their interest and assistance in this project.

I would also like to thank the following people, who were involved in the design and implementation of the ext2 filesystem kernel code and support utilities:

- `Remy Card`

Who designed, implemented and maintains the ext2 filesystem kernel code, and some of the ext2 utilities. Remy Card is also the author of several helpful slides concerning the ext2 filesystem. Specifically, he is the author of `File Management in the Linux Kernel` and of `The Second Extended File System - Current State, Future Development`.

- `Wayne Davison`

  Who designed the ext2 filesystem.

- `Stephen Tweedie`

  Who helped designing the ext2 filesystem kernel code and wrote the slides `Optimizations in File Systems`.

- `Theodore Ts´o`

  Who is the author of several ext2 utilities and of the ext2 library `libext2fs` (which I didn't use, simply because I didn't know it exists when I started to work on my project).

Lastly, I would like to thank, of-course, `Linus Torvalds` and the `Linux community` for providing all of us with such a great operating system.

Please contact me in a case of bug report, suggestions, or just about anything concerning EXT2ED.

Enjoy,

Gadi Oxman <tgud@tochnapc2.technion.ac.il>

Haifa, August 95