Reliant UNIX ONLINE Documentation

Reliant UNIX 5.44

UNIX File System (UFS)

Edition September 1997

Copyright and Trademarks

1 Introduction

At present, Reliant UNIX supports the file system types *VxFS*, *ufs*, *nfs*, and *hs*. Apart from the general descriptions, this manual explains how to organize, manage, maintain, and check the consistency of the *ufs* file system.

Information on the *VxFS* file system can be found in the "VxFS File System, System Administrator Guide" manual.Information on the *nfs* file system can be found in the "Network Administrator Guide" manual. Information on the *hs* file system is given in the "CD ROM Notes for System Administrators" manual. Other special file system types, for example *proc*, are not discussed here.

This manual is structured as follows:

- The first three chapters explain the organization of a file system, the structure of the *ufs* file system type, and the various types of file system commands.
- The section on "Administering file systems" describes the following tasks:
  - Displaying file system types with *crash*(1M)
  - Identifying file system types with *fstyp*(1M)
  - Creating file systems with *mkfs*(1M) or *newfs*(1M) (*newfs* for *ufs* file systems only)
  - Mounting file systems with *mount*(1M), *mountall*(1M), or *mountfsys*(1M)
  - Unmounting file systems with *umount*(1M), *umountall*(1M), or *umountfsys*(1M)
  - Copying file systems with *volcopy*(1M), *ufsdump*(1M)/*ufsrestore*(1M), *cpio*(1), *tar*(1), or *dd*(1)
  - Optimizing the *ufs* file system with *tunefs*(1M)
  - Outputting statistical information on file systems with *ff*(1M) or *ncheck*(1M)
  - Searching for errors in file systems with *fsdb*(1M)
  - Assigning labels to file systems with *labelit*(1M)
- The chapter on "Maintaining file systems" describes the following tasks:
  - Monitoring the disk utilization level with *df*(1M)
  - Limiting file and directory sizes with *tail*(1)
  - Determining inactive files with *find*(1)
  - Determining users with a high memory requirement with *du*(1M) or *find*(1)
  - Limiting file system reserves for users by means of the quota system
- The chapter on "Checking the consistency of file systems" describes the *fsck*(1M) utility.

The commands listed above marked with "1M" are described in detail in the "System Administrator's Reference Manual", while the commands marked with "1" are described in the "Commands User's Reference Manual".

1.1 Notational conventions

The following notational conventions are used in this manual:

Italics	File and program names, commands, options, constants, and variables in the continuous text; variables in examples and system output
`Typewriter text`	System output like error messages, news, indications, or file extracts
Bold typewriter text	User input in examples
"Double quotes"	References to other chapters or manuals, and emphasis in continuous text

ℹ	Additional information, indications, and hints
!	Warnings that you must observe under any circumstances

2 Organizing a file system

Under Reliant UNIX, a file is a byte character string without an implicit structure. Files are linked in a hierarchy of directories. A directory is another file type, which the user can read but to which the user cannot write. Only the operating system can write to directories. A file system is made up of a combination of files and directories. Figure 1. illustrates the relationship between directories and files in a Reliant UNIX file system, where the circles represent directories.

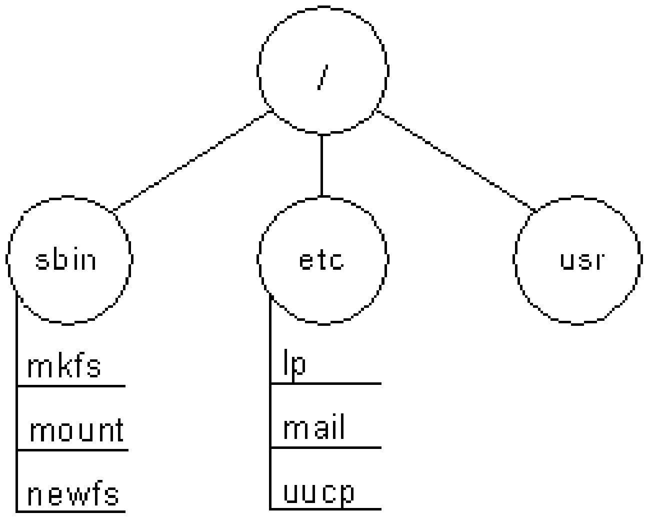

Figure 1: Reliant UNIX file system

The starting point of every Reliant UNIX file system is a directory which serves as the root of this file system. In the Reliant UNIX operating system, there is always a file system called *root*. Traditionally, the root directory of the root file system is represented by a single forward slash ("/"). The file system shown in Figure 1 is a root file system. Figure 1. illustrates the result of mounting another file system in the */usr* directory of the root file system.

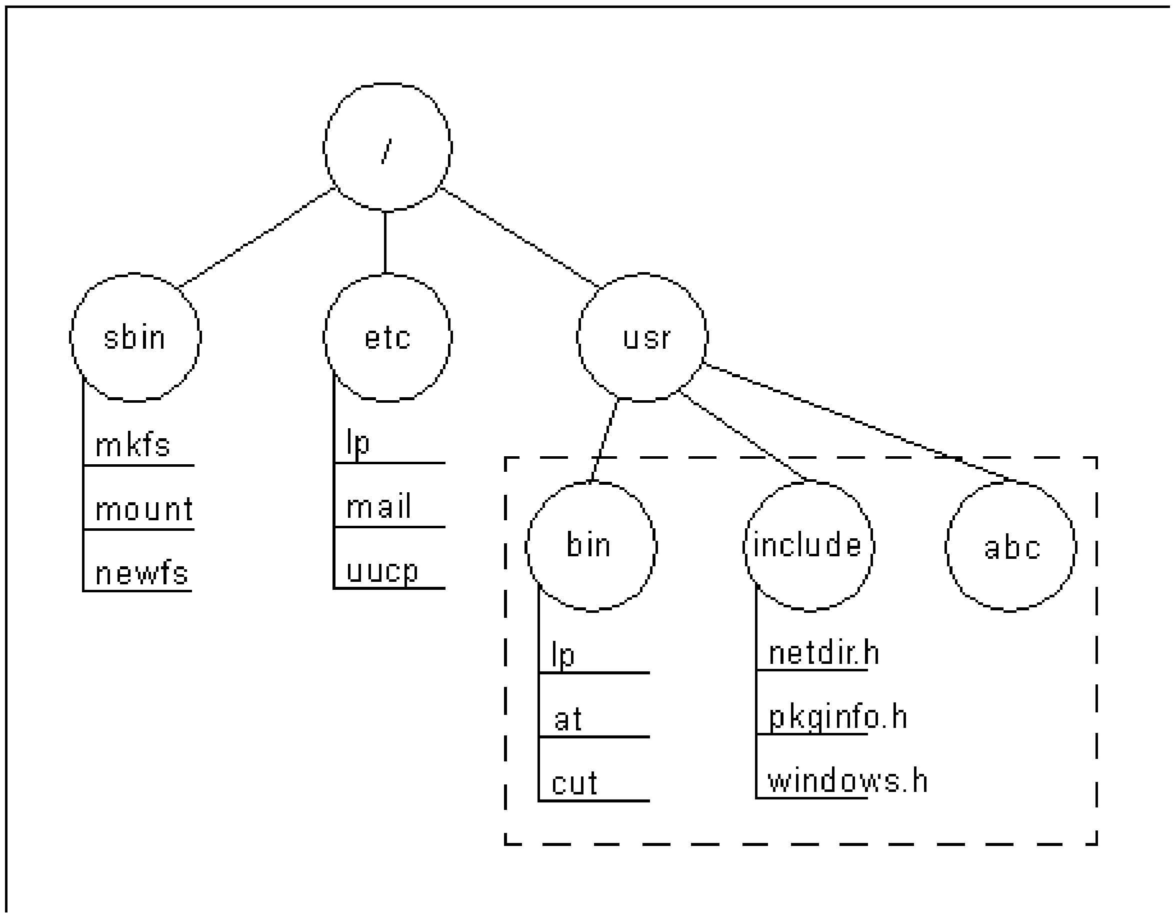

Figure 2: Adding the /usr file system

A directory such as */usr*, which is used as a link between the root file system and another mountable file system, is called an access point. This type of directory is the root of the file system descending from it. The name of this file system is the directory name; in our example, this is */usr*.

Figure 1 and Figure 2 are intended as practical representations of the file and directory structure of file systems, but are not particularly useful in illustrating how the Reliant UNIX operating system sees a file system. The chapter on the "Structure of the ufs file system type" describes how this file system type appears to the operating system.

3 Structure of the ufs file system type

With the *ufs* file system type, a magnetic disk (or piece of one or more disks) is divided into a number of cylinder groups. The individual cylinder groups are in turn divided into the following areas:

1. The load block appears only in the first cylinder group (cylinder group 0) of a file system, and occupies the first 8 Kbytes. It is reserved for procedures that are used when loading the system. If a file system is not used for loading, the load block remains empty.
2. The superblock contains all essential information on the file system:
   - Name, size, status, and access point of the file system
   - Size of the cylinder group
   - Date and time of the last modification to the superblock
   - Summary of changes in the file system (information block)

   The superblock is present in each cylinder group. It is staggered so that there is a superblock on every disk interface of a disk drive. If the first interface is not available, an alternative superblock can be called. For all interfaces, apart from the highest in a disk drive, the blocks that remain free due to staggering are reserved for data storage.

   A summary information block is stored together with the superblock. It is not duplicated, because it can be reconstructed. Instead, it is generally stored together with the first superblock in the cylinder group 0. This information block is used to register changes that occur when using the file system; it notes the number of Inodes, blocks and fragments in the file system. Storage areas that are currently not used as Inodes, indirect blocks (see description of cylinder information block) and data blocks are also identified.
3. Mapping the cylinder group involves a block that is present in each cylinder group, and which records the utilization of data blocks in the cylinders.
4. The cylinder information block contains the Inodes, which in turn contain all necessary information about the file to which they belong. Each Inode is 128 bytes long. One Inode is created for every 2 Kbytes of available storage space in a file system. This parameter can be changed when a file system is created. Each Inode contains the following information:
   - File type (normal file with user data, or program, directory, block- or character-oriented file, symbolic reference or FIFO file (named pipe))
   - Number of fixed references to the file (link count)
   - Access rights to the file
   - User number of the file owner
   - Group number to which the file belongs
   - File size in bytes
   - Twelve address fields with data block addresses (direct addressing)
   - One address field with the number of an address block for addressing data blocks (indirect addressing)
   - One address field with the number of a duplicate address block for addressing address blocks, which in turn contain data block addresses (duplicate indirect addressing)
   - Date and time of last file access
   - Date and time of last file change
   - Date and time of file creation

   Figure 3 illustrates the concatenation of address blocks, specified by the Inode.

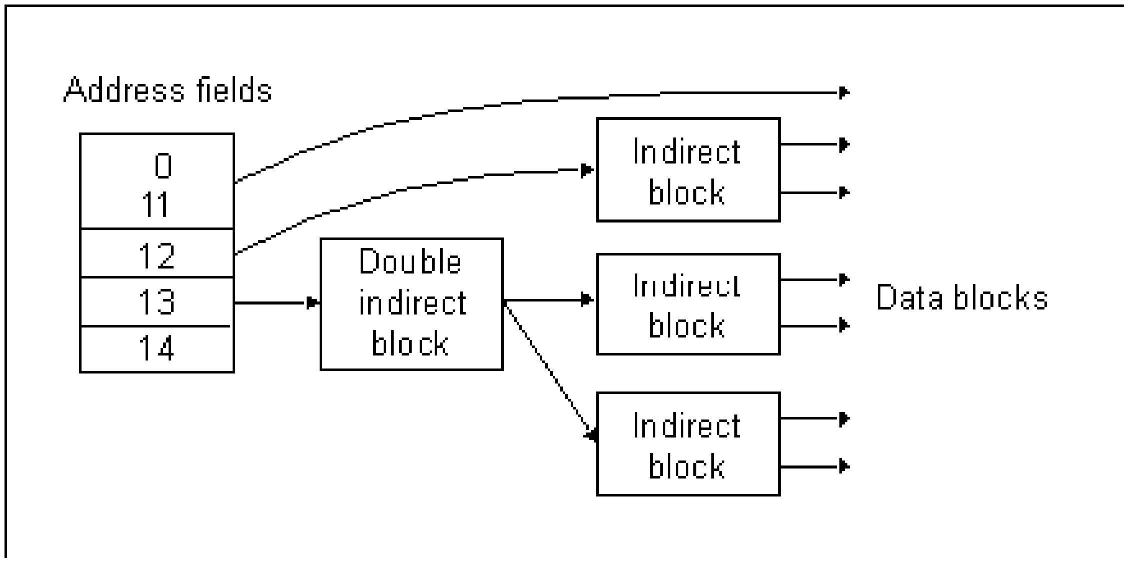

Figure 3: The file system address chain in a ufs file system

The maximum size of a *uf*s file system is 32 Tbytes, and the maximum file size is 1 Tbyte.

5. Any remaining space assigned to a file system is taken up by data blocks. The size of these data blocks is defined when a file system is created and can be 4, 8, or 16 Kbytes. Because of these considerable block sizes and due to the risk of small files wasting space, *ufs* also divides blocks into units called fragments.

   The fragment size can be a binary fraction (up to one eighth) of the block size (e.g. 1/2, 1/4, or 1/8). It must not be smaller than the physical sector size of the physical disk (e.g. 512 bytes on MP14 disks). The standard size for a block or a fragment is 16 Kbytes or 2 Kbytes.

   With a normal file, the data blocks contain the file contents. With a directory, the data blocks contain entries which specify the Inode number and the file name. *ufs* file names can be up to 255 bytes long. Each entry represents a file or a subdirectory of the particular directory.

4 File system commands

4.1 Commands for administering, maintaining, and checking file systems

The architecture of virtual file systems permits several file system types to co-exist in the system kernel. Each file system type has certain characteristics which distinguish it from other file system types. However, most file system commands offer a common interface which enables the system administrator to administer, maintain, and check the consistency of various file system types.

The following types of file system commands are available:

- Pure generic commands, which can be used on all types of file system:

  fstyp(1M)
  : identifies the file system type

  mountall(1M)
  : mounts several file systems

  umountall(1M)
  : unmounts several file systems

- Generic commands with modules specifically for the particular file system type:

  df(1M)
  : displays the disk utilization

  ff(1M)
  : displays information about a file system

  fsck(1M)
  : checks the consistency of file system

  fsdb(1M)
  : corrects errors in the file system

  labelit(1M)
  : provides labels for file systems

  mkfs(1M)
  : creates a file system

  mount(1M)
  : mounts a file system

  mountfsys(1M)
  : menu interface for *mount*(1M)

  ncheck(1M)
  : generates a list of pathnames and Inode numbers

  umount(1M)
  : unmounts a file system

  umountfsys(1M)
  : menu interface for *umount*(1M)

  volcopy(1M)
  : copies a file system

  Most generic commands can be called as follows:

  command [-F dst] [generic_options] [-o specific_options|other_options] [operands]

  The *-F* option identifies the file system type to be used.

  Generic options are interpreted both by the command module specific to the file system type, and by the generic module.

  Specific options are only interpreted by the command module specific to the file system type. They cannot be specified in conjunction with other options. With some commands, specific modules are not available for all file system types.

Other options are only interpreted by the generic module of a command.

Operands are specific to the file system type to a certain extent. They generally contain special file names, storage medium names, directory names, or access points.

- Commands that are only available for the file system type *ufs*:

  newfs(1M)
  : user-friendly front-end for *mkfs*(1M)

  ufsdump(1M)
  : saves a file system

  ufsrestore(1M)
  : restores a file system

  tunefs(1M)
  : optimizes a file system

- Apart from the commands listed above, i.e. *volcopy*, *ufsdump* and *ufsrestore*, the commands *cpio*(1), *tar*(1), *dd*(1) and *quickdd*(1M) are also available for backing up and restoring file systems.

4.2 The vfstab file

Because generic commands can be used on several types of file system, most of them require specific information, which can be specified explicitly in the command line using the *-o* option, or can be taken implicitly from the */etc/vfstab* file.

The */etc/vfstab* file contains a list of standard parameters for each file system. It is an ASCII file, which the system administrator should maintain. Each data record contains information (separated by blank lines) about a file system in the following format:

blockdev fsckdev mountpoint fstype fsckpass automnt mountopts

The individual fields are described below:

- *blockdev* – block-oriented special file for local devices, virtual disks or the name of a remote file system (e. g. *nfs*).
- *fsckdev* – character-oriented special file that corresponds to *blockdev*. If no character-oriented device is available, enter a dash.In this case, the block-oriented device is used.
- *mountpoint* – standard access point.
- *fstype* – file system type.
- *fsckpass* – flag used by *ff*, *fsck*, and *ncheck* to determine whether the file system is to be checked automatically. If this field contains a numerical entry, the file system is checked automatically. A dash prevents automatic checking of the file system.
- *automnt* – in this field, the values *yes* or *no* indicate whether the file system is to be mounted automatically with *mountall* when the system is loaded.
- *mountopts* – a list of options, separated by commas, which is used when the file system is mounted. If you do not wish to specify any options, use a dash. A list of possible options is given in the command description for *mount* (see "System Administrator's Reference Manual").

The following example shows an entry from the */etc/vfstab* file. Lines that begin with the character # are comments.

```
#□block□□□□□□□□□□□□□□□fsck□□□□□□□□□□□□□□□□□□□mount□□□□□fs□□□□fsck□□auto□□
mount
#□dev□□□□□□□□□□□□□□□□□dev□□□□□□□□□□□□□□□□□□□□point□□□□□type□□pass□□mnt□□
□opts
/dev/ios0/sdisk010s0□□/dev/ios0/rsdisk010s0□□/□□□□□□□□□ufs□□□-□□□□□-□□□□□-
/dev/ios0/sdisk000s0□□/dev/ios0/rsdisk000s0□□/oldroot□□ufs□□□-□□□□□-□□□□□ro
```

```
#□Remember□where□swap□partitions□are
/dev/ios0/sdisk000s1□□/dev/ios0/rsdisk000s1□□-□□□□□□□□□swap□□-□□□□□-□□□□□rw
#□Mount□process□subsystem□and□file□descriptor□file□systems
/proc□□□□□□□□□□□□□□□□□□□-□□□□□□□□□□□□□□□□□□□□□□□□□□/proc□□□□□proc□□-□□□□□-□□□□
□rw
/dev/fd□□□□□□□□□□□□□□□□□-□□□□□□□□□□□□□□□□□□□□□□□□□□/dev/fd□□□fdfs□□-□□□□□yes□□□rw
#□NFS□mount□mounts
```

4.3 Commands for creating and administering quota systems

In addition to the actual file system commands, commands are also available for the *ufs* file system type for creating and administering quota systems in order to limit the file system reserves for users. In a wider context, these commands can be regarded as file system commands. The following commands are available for quota systems:

edquota(1M)
  defines user quotas

quot(1M)
  displays the owner status for file systems

quota(1M)
  displays disk quotas and disk assignment for a user

quotacheck(1M)
  checks the consistency of file system quotas

quotaoff(1M)
  deactivates file system quotas

quotaon(1M)
  activates file system quotes

repquota(1M)
  displays quotas for file systems

5 Administering file systems

5.1 Displaying file system types

A list of the installed file system types is displayed using the *crash* command (see the following example):

```
# crash <<!
> vfssw
> !
dumpfile = /dev/mem, namelist = /stand/unix, outfile = stdout
> FILE SYSTEM SWITCH TABLE SIZE = 11
SLOT   NAME     FLAGS
□□□1□□□fdfs□□□□□□□82001
□□□2□□□fifofs□□□□□42001
□□□3□□□hs□□□□□□□□□1
□□□4□□□namefs□□□□□82000
□□□5□□□nfs□□□□□□□□21000
□□□6□□□proc□□□□□□□82001
...
> #
```

Apart from the recognized file system types, *crash* also lists internal file system types that do not have a user interface.

5.2 Identifying file system types

Most file system commands require that the file system type be specified in the command line or in the */etc/vfstab* file. If the system administrator specifies the wrong type, the commands will detect this and react with an error. However, it is still important to specify the correct type, since file systems can be destroyed if a command fails to recognize a system administrator error and if an operation that can only be executed on one system is executed on another.

The file system type can be identified with the *fstyp* command, regardless of whether or not the file system is mounted. The command is called as follows:

fstyp *special_file*

Example:

```
# fstyp /dev/ios0/sdisk001s0
```

Output is written to the screen. If the type cannot be specified or cannot be specified uniquely, the message unknown_fstyp (no matches) or unknown_fstyp (multiple matches) is output.

5.3 Creating a file system

File systems are created using the *mkfs* command. This command requires detailed user input and assumes a thorough knowledge of the file system to be created, as well as the geometry of the magnetic disk.

The *newfs* command is used for the file system type *ufs*. This command is a user-friendly front-end to *mkfs*, and simplifies creation of the file system.

Be very careful when selecting the physical area or the virtual disk where you want to create or recreate a file system, because all the data in this area will be deleted during the creation process. If you want to recreate a file system you should first create a backup of the old system.

Remember that a file system must be mounted after it is created so that it can be accessed by all users.

5.3.1 Creating a file system with mkfs

A file system with a root and lost+found directory is created with the *mkfs* command. The number of Inodes is calculated as a function of the file system size.

To create a file system, the *mkfs* command can be called in the following format:

mkfs [-F *dst*] [-o *specific_options*] *special_file file_system_size*

If there is no appropriate entry in the */etc/vfstab* file, you must specify the file system type in the command line with the *-F* option.

The *-o* option can be used to specify parameters specific to the file system type, which suit the geometry of the disk type. The required parameter values can be determined, for example, with the *dkpart* and *setinfo* commands. If the *-o* option is not specified, default values are used.

The *file_system_size* parameter specifies the number of sectors in the file system.

The following example illustrates the call for a *ufs* file system:

```
mkfs -F ufs -o nsect=26,ntrack=15,apc=6 /dev/ios0/rsdisk000s2 1198860
```

5.3.2 Creating a ufs file system with newfs

File systems of type *ufs* can be created with the *newfs* command. File systems of this type have many parameters that refer to the geometry of the disk. *newfs* automates the task of defining values for these parameters.

To create a file system, the *newfs* command can be called in the following format:

newfs [*specific_options*] *special_file*

If specific options are given, the parameters are not defined automatically. These options are the same as those specified with the *-o* option in the *mkfs* command.

The *special_file* parameter identifies a character-oriented device.

The following example illustrates the call for a ufs file system:

```
newfs /dev/ios0/rsdisk000s2
```

5.3.3 Recreating a file system

To change the file system-specific parameters of an existing file system, proceed as follows:

Unmount the old file system (this is only possible if you want to use the *ufsdump* and *ufsrestore*) commands:

```
umount /opt
```

Save the old file system with the *cpio* or *tar* command. For *ufs* file systems, you can also use the *ufsdump* command:

```
ufsdump 0uf /dev/ios0/rstape004 /dev/ios0/rsdisk000s2
```

Create the new (empty) file system with the modified parameters:

```
newfs -b 8192 -f 1024 /dev/ios0/rsdisk000s2
```

Mount the new file system (specify block-oriented special file):

```
mount -F ufs /dev/ios0/sdisk000s2 /opt
```

Copy the old file system to the new file system with the *cpio* or *tar* command. If you used the *ufsdump* command to make a backup, you can recreate the file system with the *ufsrestore* command:

```
ufsrestore xf /dev/ios0/rstape004 /dev/ios0/rsdisk000s2
```

5.4 Mounting and unmounting file systems

A file system must be mounted so that it can be accessed by all users. When the system is powered up, all the file systems entered in the */etc/vfstab* file, and for which the *automnt* field is set to *yes*, are mounted automatically. All mounted file systems are entered in the */etc/mnttab* file automatically. When the system is powered down, all the file systems entered in the */etc/mnttab* file, with the exception of / (Root), */proc*, and */dev/fd*, are unmounted.

5.4.1 Mounting file systems with mount

To mount a file system, the *mount* command is called in the following format:

mount [-F *dst*] [-o *specific_options*] *special_file access_point*

If there is no appropriate entry in the */etc/vfstab* file, the file system type must be specified in the command line with the *-F* option.

Parameters specific to the file system can be specified with the *-o* option. Possible parameters are given in the command description for *mount*.

If there is no appropriate entry in the */etc/vfstab* file, the name of the block-oriented special file and the access point must be specified; otherwise, use one these entries is sufficient.

Example of mounting a file system with *mount*:

mount -F ufs /dev/ios0/sdisk000s2 /opt

To mount several file systems, the *mount* command can be called in the following format:

mount [-F *dst*] -a

If the command is called in this format, all file systems of the specified type, entered in the */etc/vfstab* file, and for which the *automnt* field is set to *yes* are mounted. If no file system type is specified, all file systems for which the *automnt* field is set to *yes* are mounted.

5.4.2 Mounting file systems with mountall

To mount several file systems, the *mountall* command can be called in the following format:

mountall [-F *dst*] [/etc/vfstab|-]

All file systems of the specified type, entered in the */etc/vfstab* file, and for which the *automnt* field is set to *yes* are mounted. If no file system type is specified, all file systems for which the *automnt* field is set to *yes* are mounted. If a dash is specified instead of the file name, standard input is read. Standard input must be in the same format as the entries in the */etc/vfstab* file.

In the following example, all file systems of type *ufs* entered in the */etc/vfstab* file for which the *automnt* field is set to *yes* are mounted.

mountall -F ufs

5.4.3 Mounting file systems with mountfsys

The *mountfsys* command is a menu interface to the *mount* command.

5.4.4 Unmounting file systems with umount

To unmount a file system, the *umount* command can be used in the following format:

umount [-F *dst*] *special_file*|*access_point*

The type of the file system to be mounted can be specified with the *-F* option.

Example of unmounting a file system with *umount*:

umount /opt

To unmount several file systems, the *umount* command can be called in the following format:

umount [-F *dst*] -a

If the command is called in this format, all file systems of the specified type, and entered in the */etc/mnttab* file are unmounted. If no file system type is specified, all the file systems in the */etc/mnttab* file are unmounted.

The file systems / (Root) and /proc cannot be unmounted with mount -a.

5.4.5 Unmounting file systems with umountall

To unmount several file systems, the *umountall* command can be called in the following format:

umountall [-F *dst*]

All the file systems of the specified type, and entered in the */etc/mnttab* file are unmounted. If no file system type is specified, all the file systems entered in the */etc/mnttab* file are unmounted.

The file systems / (Root), /proc, /var, /usr, and /dev/fd cannot be unmounted with umountall.

5.4.6 Unmounting file systems with umountfsys

The *umountfsys* command is a menu interface to the command *umount*.

5.5 Copying file systems

The commands described below can be used to copy file systems. The examples used assume the following:

1. The file system is mounted in */old*, and the special file is called *rsdisk000s0*.
2. The file system is to be copied to *rsdisk001s0* and mounted in *new*.

Because some of the commands require that the old and/or new file systembe mounted during copying, there may be inconsistencies between the old file system and the copy, if one of the file systems is accessed at the time of copying (see the individual descriptions).

5.5.1 Copying file systems with volcopy

The physical copy of a file system can be created with the *volcopy* command. The character-oriented device must be specified. If *volcopy* is used, the new file system has the same structure as the old file system. It is not possible to convert file system parameters (e.g. change the logical block size) with *volcopy*.

The old file system must be mounted during copying. The new file system must be unmounted during copying.

The command can be called in the following format:

volcopy [-F *dst*] *file_system_name source_special_file source_device_name target_device_file target_device_name*

If there is no appropriate entry in the */etc/vfstab* file, the file system type must be specified in the command line with the *-F* option.

The file system name is the same as that specified with *labelit* (see the Section "Specifying identifiers for file systems"). If no file system name is specified, an empty string ("") must be specified here.

A dash can be specified instead of the physical data carrier name for the source and target devices. The dash indicates that the existing data carrier name is to be used. The section on "Specifying Identifiers for File Systems" describes how to assign physical names.

Example of copying a file system with *volcopy*:

```
mount /old
umount /new
volcopy -F ufs old /dev/ios0/rsdisk000s0 - /dev/ios0/rsdisk001s0 -
mount -F ufs /dev/ios0/sdisk001s0 /new
```

5.5.2 Copying file systems with ufsdump/ufsrestore

The logical copy of a file system of type *ufs* can be created with the commands *ufsdump* and *ufsrestore*. *ufsdump* saves a file system, and *ufsrestore* then restores thatfile system.

The old file system must be unmounted and the new (empty) file system created and mounted before copying begins.

To copy a file system, the ufsdump and *ufsrestore* commands can be called in the following format:

ufsdump 0f - *source_special_file* | (cd *new_access_point*; ufsrestore xf -)

Here, the character "|" is a pipe, and not a metacharacter for the "exclusive or".

ufsdump uses backup levels *0* through *9*. This means that both entire file systems and individual files can be backed up. Backup level *0* saves entire file systems.

The *f* option is used to replace the default backup medium with the one specified. If a dash is specified as the backup medium, *ufsdump* writes to standard output, and *ufsrestore* reads from the standard input. Thus, both commands can be used in a pipe to back up and restore a file system.

Example of copying a file system with *ufsdump/ufsrestore*:

```
umount /old
newfs /dev/ios0/rsdisk001s0 /new
mount -F ufs /dev/ios0/sdisk001s0 /new
ufsdump 0f - /dev/ios0/rsdisk000s0 | (cd /new; ufsrestore xf -)
```

5.5.3 Copying file systems with cpio

The logical copy of a file system can be created with the *cpio* command. Because the command reads from the standard input, it is generally used in a command pipe, where a selection program functions as a driver.

The old file system must be mounted during copying. The new (empty) file system must be created and mounted before copying.

The command can be called in the following format:

cpio -p[*options*] *new_access_point*

At least the following two options should be specified:

- The *d* option, which ensures that the necessary directories are created automatically.
- The *m* option, which ensures that the last modification date for the individual files does not change.

Example of copying a file system with *cpio*:

```
mount /old
newfs /dev/ios0/rsdisk001s0 /new
mount -F ufs /dev/ios0/sdisk001s0 /new
cd /old
find . -print | cpio -pdm /new
```

5.5.4 Copying file systems with tar

The logical copy of a file system can be created with the *tar* command. *tar* can be used in a pipe to copy file systems, which means an external data carrier is not required.

The old file system must be mounted during copying. The new (empty) file system must be created and mounted before copying.

The command can be called in the following format:

(cd *old_access_point*; tar cf - .) | (cd *new_access_point*; tar xf -)

Here, the character "|" is a pipe, and not a metacharacter for the "exclusive or".

The *f* option is used to replace the default special file to which data is written or from which data is read with the one specified. If a dash is specified instead, data is output to standard output or read from standard input. Thus, *tar* can be used in a pipe to copy a file system.

Example of copying a file system with tar:

```
mount /old
newfs /dev/ios0/rsdisk001s0 /new
mount -F ufs /dev/ios0/sdisk001s0 /new
(cd /old; tar cf - .) | (cd /new; tar xf -)
```

5.5.5 Copying file systems with dd

A physical copy of a file system can be created with the *dd* command. When copying, it is important to remember that only block sizes that correspond to the physical block size of the hard disk (or a multiple thereof) can be used to describe character-oriented devices. For this reason, it is inadvisable to use *dd* to copy file systems from one hard disk to another, if the disks have different physical block sizes.

If *dd* is used, the new file system has the same structure as the old file system. It is not possible to convert the file system parameters (e. g. change the logical block sizes) with *dd*.

Before copying, both the old and the new file system must be unmounted.

The command can be called in the following format:

dd [*option* ...]

Of all the options available, only the following three are explained here:

- The *if=source_file* option specifies the source device from which *dd* is to read. If this option is not specified, *dd* reads from standard input.
- The *of=target_file* option specifies the target device to which *dd* is to write. If this option is not specified, *dd* writes to standard output.
- The *bs=blocksize* option specifies the size of the input and output blocks in bytes.

Example of copying a file system with *dd*:

```
umount /old
umount /new
dd if=/dev/ios0/rsdisk000s0 of=/dev/ios0/rsdisk001s0 bs=32k
mount -F ufs /dev/ios0/sdisk001s0 /new
```

5.6 Optimizing file systems of type ufs

The *tunefs* command can be used to change the dynamic parameters of a *ufs* file system. The file system must be unmounted before it is optimized.

ufs file systems should only be optimized by those system administrators who have a thorough knowledge of the organization of such file systems.

The command can be called in the following format:

tunefs [*options*] *special_file*|*file_system_name*

The *-a decimal_value* option specifies the maximum number of contiguous blocks that can be transferred before disk access is interrupted. This value is "1" for RM200, RM300, RM400, and RM600 type systems.

The *-d decimal_value* option specifies the length of time in milliseconds between the end of a transfer and the start of a new transfer on the same disk. This value depends on the disk type. If the specified value is too low or too high, a complete disk rotation takes place before the new transfer.

Optimization should not be attempted if it affects the time. The read/write cache of current disks is geared for optimum times.

The *-e decimal_value* option specifies the maximum number of blocks that can be assigned to a file in a cylinder group before additional blocks must be assigned in another cylinder. The typical value is approximately 25% of the blocks in a cylinder group. This restriction means that the average access time is shorter for smaller files.

For file systems containing only large files, this parameter should be set at a higher value.The *-m decimal_value* option specifies the percentage of storage space not available to ordinary users. Once the file system capacity reaches this threshold value, only authorized users can assign storage space for files. The default value is 10%. In the case of large magnetic disks, a smaller value should be selected. It is also important to ensure that a smaller threshold value does not affect performance.

The *-o* {*space* | *time*} option changes the optimization strategy, which defines the area in which optimization is necessary. In general, the access time is to be optimized. Storage space is optimized only if the file system is very full. The optimization strategy can be defined when the file system is created. If this is not the case, the strategy is generally geared towards the access time.

Optimization should not be attempted if it affects the time. The read/write cache of current disks is geared for optimum times.

Example of optimizing a magnetic disk of type MP65 (1.3 Gbytes net):

```
tunefs -d 2 -m 5 /dev/ios0/rsdisk000s0
```

5.7 Displaying statistics on file systems

The *ff* and *ncheck* commands display statistics on the specified file systems.

The commands can be called in the following format:

ff [-F *dst*] *special_file ...*
ncheck [-F *dst*] [*special_file ...*]

If there is no appropriate entry in the */etc/vfstab* file, the file system type must be specified in the command line with the *-F* option.

If *ncheck* does not specify a special file, a list of all the devices that have a numerical entry in the *fsckpass* field of the */etc/vfstab* file are displayed.

Both commands specify the I number and the pathname of the specified devices. If the generic options *-s* and/or *-u* are specified with the *ff* command, the file size (in bytes) and/or the user ID of the owner are also displayed.

Example of the output of status information with *ff*:

```
ff -F ufs -s -u /dev/ios0/rsdisk000s3
/dev/ios0/rsdisk000s3:
3□□□□□□□/lost+found/.□□□□16384□□root
2048□□□□/add-on/.□□□□□□□□□□512□□bin
4□□□□□□□/adm□□□□□□□□□□□□□□□□□□0□□root
4096□□□□/admin/.□□□□□□□□□□□512□□bin
32769□□□/bin/.□□□□□□□□□□□□6656□□bin
34817□□□/ccs/.□□□□□□□□□□□□□512□□bin
...
```

5.8 Troubleshooting program for file systems

With the *fsdb* command, a corrupted file system can be corrected interactively after a system crash.

There are different troubleshooting options for the different file system types. *fsdb* contains routines for checking Inodes and block addresses, as well as routines for converting Inodes and block numbers into disk addresses.

fsdb is an extensive troubleshooting program, and is intended for experienced system administrators only. Additional information on troubleshooting options and on working with this command is given in the command description in the "System Administrator's Reference Manual".

5.9 Specifying identifiers for file systems

Using the command *labelit*, a file system can be assigned any file system name and data carrier name.

The command can be called in the following format:

labelit [-F *dst*] *special_file* [*file_system_name data_carrier_name*]

If there is no appropriate entry in the */etc/vfstab* file, the file system type must be specified in the command line with the *-F* option.

File system names and data carrier names can be up to six characters long. Existing names can be deleted by entering an empty string (""). The file system must be unmounted.

Example of specifying identifiers with *labelit*:

```
labelit -F ufs /dev/ios0/rsdisk000s2 file1 disk1
```

6 Maintaining file systems

Once a system has been created and is available, it must be maintained regularly, to ensure that performance is satisfactory and inconsistencies are prevented. This chapter describes how to maintain a file system so that performance is kept at a satisfactory level. Consistency is discussed in the Chapter "Checking the consistency of file systems".

There are five tasks that the system administrator should perform regularly to determine whether file system performance is satisfactory. The aim of each of these tasks is to ensure that storage space on the magnetic disk is not reduced to a level where system performance is affected.

1. Monitoring disk utilization
2. Restricting the size of files and directories
3. Identifying inactive files
4. Identifying users who require a lot of disk space
5. Restricting file system reserves for users

6.1 Monitoring disk utilization

The *df* command can be used to monitor disk utilization at any time. It is advisable to check the disk capacity every day until a schema has been developed.

To monitor the disk storage space, the command can be entered in the following format:

df [-F *dst*] -k [*directory*|*special_file*]

If there is no entry in the */etc/vfstab* file, or if the command is used on an unmounted file system, the file system type must be specified in the command line with the *-F* option.

The *-k* option outputs the following values:

filesystem
: Name of the file system

Kbytes
: Gross storage space in Kbytes

used
: Available storage space (occupied) in Kbytes

avail
: Available storage space (free) in Kbytes

capacity
: Percentage of occupied storage space in relation to the total amount of storage space available

mounted on
: Access point

The difference between the gross storage space and the sum of the occupied and free storage space is calculated from the value specified when the file system is created (*minfree* parameter in *mkfs*) or optimized (*-m* option in *tunefs*).

If neither a directory nor a special file are specified, *df* provides information on all the mounted file systems (which can be restricted with the *-F* option).

Example of monitoring disk utilization with *df*:

```
df -F ufs -k /dev/ios0/sdisk000s3
filesystem□□□□□□□□□□□□□□□□□kbytes□□□□□used□□□avail□□capacity□□mounted□on
/dev/ios0/sdisk000s3□□□□□□127026□□□112682□□□□1640□□□□□99%□□□□/usr
```

6.2 Restricting the size of files and directories

Almost every system that is used on a daily basis contains files and directories which are constantly increasing in size, for example:

File	Usage
/var/adm/wtmp	System login log
/var/adm/sulog	*su* command log
/var/cron/log	Log of */usr/sbin/cron* actions
/var/help/HELPLOG	*/usr/bin/help* actions
/var/spell/spellhist	Words that *spell* could not find

The frequency with which files are checked depends on how often the system is used and how critical the problem of storage space is. The *tail* command is used to restrict the size of files.

In the following example, the */var/adm/sulog* file is shortened to the last 50 entries:

```
tail -50 /var/adm/sulog > /var/tmp/sulog
mv /var/tmp/sulog /var/adm/sulog
```

6.3 Identifying inactive files

Cleaning up file systems that are heavily loaded involves identifying files that have not been used recently. These files can be identified using the *find* command.

In the following example, normal files that have not been modified in the last 60 days are searched for, and output is redirected to a temporary file. This process runs in the background. This precaution is useful if complete output is required.

```
find /home -type f -mtime +60 > /var/tmp/deadfiles &
```

6.4 Identifying users who require a lot of space

The *du* and *find* commands can be used to identify users who require a lot of storage space.

The *du* command outputs the number of blocks of all the files or directories specified in the command line. For example,

```
du /home
```

displays the number of blocks of all directories in the file system */home*.

The *find* command can be used to locate files that exceed a particular size.

The following example specifies the pathnames of all directories and files in the */home* file system, that are greater than 10 blocks (each with 512 bytes). The block size of 512 bytes is hardwired in the *find* command and is independent of the file system parameters.

```
find /home -size +10 -print
```

6.5 Restricting file system reserves for users

Using a quota system, it is possible to restrict user access to the two main reserves of a file system, i.e. Inodes and data blocks. User quotas consist of two limits, an absolute (hard) limit and a relative (soft) limit, both of which can be specified for either of the main reserves.

A user cannot exceed the absolute limit. Any attempt to do so will fail and the user will receive a message indicating that no more space is available.

A user can exceed a relative limit for a specified length of time in order to use file system reserves. There is no warning when a user exceeds the relative limit. If the user does not reduce the reserves occupied before the time limit expires, any further attempt to obtain additional file system reserves will fail. The user receives error messages indicating that the file system is full. These messages continue to appear until the user is once more within the appropriate limit. The current value for disk assignment and disk quotes can be ascertained using the command (see below).

The absolute and relative limits are defined individually for each user on the different *ufs* file systems. Time limits are defined for individual file systems, which means that they are valid for every file system user.

6.5.1 Defining user quotas

This section describes the steps involved in defining user quotas.

Before quotas are defined for the first time in a file system, the following preparatory steps must be taken:

1. Mount the file system on which quotas are to be defined
2. Change to the access point of the file system
3. Create an empty file called *quotas* (e. g.)
   > quotas

Before quotas are defined in a file system, that file system should be checked for inconsistencies with *quotacheck*. The file system must be mounted before it can be checked and no applications can run on the file system while it is being checked. If the file system type is not *ufs*, the message "cannot check" is output.

By running *quotacheck*, the *quotas* file undergoes a plausibility check, and the quotas are synchronized with the current file system status and updated. If the *-v* option is used, the modified quotas are output for each user if inconsistencies are found, e. g.:

```
quotacheck -v /home
*** Checking quotas for /dev/ios0/rsdisk000s5 (/home)
hjk□□□□□□□□fixed:□□files 0 -> 2□□blocks 0 -> 8
```

To define user quotas, the *edquota* command can be called in the following format:

edquota *user_name* ...

When *edquota* is called, a temporary file is created for each specified user. This file contains the current quotas for the user, and for all the mounted *ufs* file systems for which a *quotas* file exists. An editor is then called for each temporary file in turn, so that you can define new quotas or modify existing ones. When exiting the editor, *edquota* modifies the quota files according to the temporary files.

A temporary file created by *edquota* to define limits contains individual lines that are structured as follows:

```
fs /home blocks (soft = 0, hard = 0) inodes (soft = 0, hard = 0)
```

To define a length of time which should apply when the relative limit is exceeded, the command \f4edquota\f1 can be used in the following form:

edquota -t

When *edquota -t* is called, a temporary file is created that contains the current time limits for every mounted *ufs* file system for which a *quotas* file exists. An editor is then called so that you can define or modify time limits for blocks and Inodes. When exiting the editor, *edquota* modifies the quota files according to the temporary file.

The temporary file created by *edquota* in order to define time limits contains individual lines structured as follows:

```
fs /home blocks time limit = 1 day, files time limit = 1 day
```

When defining time limits, you can use the following units of time:

sec(onds), min(untes), hour(s), day(s), week(s), month(s)

If the time limits are set to "0", default values apply, which are defined in the */usr/include/sys/fs/ufs_quota.h* file with the constants *DQ_BTIMELIMIT* and *DQ_FTIMELIMIT*.

If quotas are defined for a file system entered in the */etc/vfstab* file, enter *rq* in the *mntopts* field for the file system. If this field contains *rw* in the table, it must be replaced with *rq*.

6.5.2 Activating/deactivating user quotas

The commands *quotaon* and *quotaoff* can be used to activate and deactivate the quotas defined for the individual files systems. The file systems must be mounted when the quotas are activated or deactivated.

The commands can be called in the following format:

quotaon *file_system* ...
quotaoff *file_system* ...

To activate or deactivate quotas, the commands can also be called in the following format:

```
quotaon -a
quotaoff -a
```

With the *-a* option, quotas are activated or deactivated for all the mounted file systems that have the abbreviation *rq* in the *mnopts* field of the */etc/vfstab* file.

6.5.3 Displaying user quotas

User quotas can be displayed using the *quota* and *repquota* commands.

The commands can be called in the following format:

quota -v *user_name*
repquota *file_system* ...
repquota -a

The *quota* command displays the disk assignment and disk quota for a user. With this command, users can receive a report about their own quotas. An authorized user can also request a report about quotas for other users.

The *repquota* command displays the disk assignment and disk quotas for file systems. The current number of files and the total occupied storage space in Kbytes is output for each user. All the quotas calculated with the *edquota* command are also displayed. The *-a* option creates a report about all the mounted file systems that have the abbreviation *rq* in the *mnopts* field of the */etc/vfstab* file.

6.5.4 Displaying the owner status for file systems

The owner status for file systems can be displayed with the *quot* command.

The command can be called in the following format:

quot [*options*] [*file_system*]

Of all the options available, only the following two are explained here: The *-f* option displays the amount of space (in Kbytes) that each user is occupying in the particular file system, and also displays how many files belong to each user. The *-a* option creates a report about all the mounted file systems. If a file system is specified, this specification is ignored.

7 Checking the consistency of file systems

Every time the Reliant UNIX operating system is booted, a plausibility check should be performed for all file systems using the *fsck -m* command. This check is performed automatically as part of the load procedure when the system is booted. The check outputs a return value for each file system, which specifies whether or not corrections are required.

If inconsistencies are detected, corrections should be made before the file systems are mounted. Although inconsistencies are not that common in the file system, they should be taken seriously. Considerable emphasis is placed on the checking of file systems, because errors that go unchecked can result in extensive loss of data.

7.1 The fsck utility

The *fsck* utility is a program for checking and repairing file systems. *fsck* works with several file system processes. Each process calls another phase of the program to check Inodes, pathnames, concatenation, reference counters, and cylinder groups, and also to delete data. Each phase reports any errors that it discovers. *fsck* outputs a message for every inconsistency that is corrected. This message provides information on how the corrections were made and on the file system in which the corrections were made. Once a file system is corrected successfully, *fsck* outputs the number of files, occupied blocks, and free blocks.

With the exception of the root file system, a file system should be unmounted when being checked. If this is not possible, it is important to ensure that the system is not active and that it is powered up immediately after a critical file system is checked. The root file system should only be checked if the system is in single user mode and no other activity is running. The system should be then be reloaded immediately.

fsck can be executed interactively and non-interactively (*-p* option).

In interactive mode, the system administrator can decide whether the inconsistencies found are to be rectified or ignored; a prompt is output asking the system administrator to confirm whether or not a correction is to be implemented. It is important to remember that some corrections can result in the loss of data. Diagnostic output can be used to determine how much data may be lost and how important that data is. A correction generally waits until the user answers with *yes* or *no*. If the user does not have write access, *fsck* sets the response to *no* by default.

It is possible to instruct *fsck* to ignore certain inconsistencies because the error is so critical that you want to remove it yourself, or you want to revert to an earlier version of the file system.

Whatever you decide, you should not ignore any inconsistencies that are reported by *fsck*. Inconsistencies in the file system do not correct themselves and become even worse if ignored.

In non-interactive mode, any inconsistencies that are discovered are automatically repaired where possible. If a file system is severely corrupted, *fsck* may fail to function. Automatic repairs may result in considerable loss of data. In this case, *fsck* suggests that you carry out the check once more in interactive mode.

To implement a plausibility check, you can call *fsck* in the following format:

fsck [-F *dst*] [-m] [*special_file* ...]

To implement a file system correction, you can call *fsck* in the following format:

fsck [-F *dst*] [*generic_options*] [-o *specific_options*] [*special_file* ...]

If there is no appropriate entry in the */etc/vfstab* file, the file system type must be specified in the command line with the *-F* option.

If no special file is specified, the command is executed for all character-oriented devices of a file system type that is entered in the */etc/vfstab* file and has a numerical entry in the *fsckpass* field.

The *-m* option checks file systems but does not correct them. This option ensures that file systems are suitable for mounting.

Of all the options available, only the following three are explained here:

- With the *-p* option, *fsck* performs file system corrections in non-interactive mode. Inconsistencies that are considered harmless, and can be corrected without confirmation from the system administrator, are corrected in this mode.
- The *-y* or *-n* option specifies that all *fsck* queries are answered automatically with *yes* or *no*. The *-y* option turns off interactive mode.

The following specific options are available for the file system type *ufs*:

- With the *b=block_number* option, the specified block is used as an alternative superblock for the file system. Block 32 always serves as an alternative for the superblock.
- With the *r* option, a file system is checked but inconsistencies are not removed.
- With the *w* option, all *ufs* file systems that have the abbreviation *rw* in the *mountopts* field of the */etc/vfstab* file are checked.

The following example illustrates how *fsck* is called to check the file system *home2*. No options are specified. The system response indicates that no inconsistencies were found. At the end of every phase, a message is output. A summary line containing the number of used files (Inodes), as well as the occupied and free blocks, is also displayed at the end of a phase.

```
fsck -F ufs /dev/ios0/rsdisk10s6
** /dev/ios0/rsdisk10s6
** Last mounted on /home2
** Phase 1 – Check Blocks and Sizes
** Phase 2 – Check Pathnames
** Phase 3 – Check Connectivity
** Phase 4 – Check Reference Counts
** Phase 5 – Check Cyl Groups
18023 files, 82738 used, 92629 free (5005 frags, 10953 blocks, 29% fragmentation)
```

7.2 Checking file systems of type ufs

This section describes how to use *fsck* in *ufs* file systems. It is assumed that \f4fsck\f1 is executed interactively and that all possible errors can be detected. If an inconsistency is discovered in this mode, it is reported so that a suitable correction can be selected.

Before describing the phases of fsck and the associated messages, this section discusses the various consistency checks that apply for each component in a *ufs* file system.

7.2.1 Checking ufs file system components with fsck

7.2.1.1 Superblock

The most vulnerable object in a file system is the information block of the superblock, because it changes each time the blocks or Inodes of a file system change. Inconsistencies are generally the result of an incorrect system abort. The following information about the superblock is checked for inconsistencies:

- File system size

  The size of the file system must be greater than the number of blocks occupied by the superblock and the Inode list. Although there is no way of controlling these sizes precisely, \f4fsck\f1 can check that they are within acceptable limits. All other file system checks assume that these sizes are exact. If \f4fsck\f1 discovers inconsistencies in the statistics parameters of the default superblock, it asks the user to specify the position of an alternative superblock.

- List of free blocks

  fsck checks that the blocks marked as free in the block schema of the cylinder group are not occupied by any file. Once all the blocks have been checked, *fsck* specifies whether the number of free blocks plus the number of blocks occupied by Inodes is the same as the total number of blocks in the file system. If there is an inconsistency in the block assignment schema, *fsck* recreates them using the list of reserved blocks that it compiled.

- Number of free blocks

  The information block in the superblock contains the total number of free blocks in the file system. *fsck* compares this number to the number of free blocks that it finds in the file system. If the numbers do not correspond, *fsck* replaces the incorrect value in the information block with the actual number of free blocks.

- Number of free Inodes

  The information block contains the total number of free Inodes in the file system. *fsck* compares this number with the number of free Inodes that it finds in the file system. If the numbers do not correspond, *fsck* replaces the incorrect value in the information block with the correct number of free Inodes.

7.2.1.2 Inodes

The list of Inodes in the file system is checked entry by entry from Inode 2 to the last Inode in the file system (Inode 0 does not exist; Inode1 is not used and reserved). The following information about Inodes is checked for inconsistencies:

- Format and type

  Each Inode contains a mode word, which describes the type and status of the Inode. There are six Inode types: normal file, directory, symbolic reference, block- or character-oriented file, and named pipe. An Inode can be in one of three statuses: not assigned, assigned, and neither/nor. The last status means that the Inode was formatted incorrectly. This happens when defective data is entered in the Inode list. In this case, the only course open to *fsck* is to delete the Inode.

- Number of references

  Each Inode counts the total number of directory entries which refer to it. *fsck* checks the number of references for each Inode, starting with the root and continuing through the entire directory structure of a file system. The actual number of references for each Inode is calculated during the search. If the stored number of references is not zero and the actual number of references is zero, there is no directory entry for the Inode. If this is the case, *fsck* enters the isolated file in the *lost+found* directory. If the stored number of references and the actual number are not zero, and not equal to each other, it may be that a directory entry was added or deleted and the Inode was updated. If this is the case, *fsck* replaces the incorrect number of references with the correct value. Each Inode contains a list or a list pointer (indirect blocks) of all the blocks occupied by the Inode. Because indirect blocks belong to an Inode, inconsistencies in an indirect block have an immediate effect on its Inode.

- Duplicate blocks

  fsck compares each block number occupied by an Inode with a list of assigned blocks. If a block number is already occupied by another Inode, this block number is added to a list of duplicate blocks. Otherwise, the list of assigned blocks is updated to include the block number. If there are duplicate blocks, *fsck* goes through the Inode list for a second time to find the Inode of the duplicated block. This is necessary to check that the contents of the files associated with the Inodes are correct, and thereby locate and delete the faulty Inode. In this case, *fsck* offers both Inodes to the user, who must then decide which is to be kept and which is to be deleted. In this situation, the Inode with the oldest modification date is generally invalid and should be deleted.

- Incorrect block numbers

  fsck checks the area of each block number occupied by an Inode. If the block number is less than the first data block, or greater than the last data block of a file system, the block number is incorrect. A large number of the incorrect blocks in an Inode are normally caused by an indirect block that is not written in the file system. This can only happen after a hardware failure. If an Inode contains incorrect block numbers, *fsck* asks the user to delete them.

- Inode sizes

  Each Inode indicates the number of data blocks that it contains. The actual number of data blocks is the sum of allocated blocks and indirect blocks. *fsck* calculates the actual number of data blocks and compares this number with the number of blocks occupied by the Inode. If an Inode contains an invalid number, *fsck* asks the user to correct this. Each Inode contains a 32-bit size field, which in turn contains the number of bytes of the file belonging to the Inode. The consistency of the size field can be checked roughly by calculating the maximum number of blocks associated with the Inode using the size field, and then comparing the expected block number with the actual number of blocks occupied by the Inode.

7.2.2 Inode data

An Inode can refer directly or indirectly to three types of data block. Complete blocks that are referred to must be of the same type. The three types of data block are: normal data blocks, data blocks for symbolic references, and directory data blocks. Normal data blocks store information in a file; data blocks for symbolic references contain the pathname saved in a reference; directory data blocks contain directory entries. *fsck* can only check the validity of the directory data blocks.

7.2.3 Directory data blocks

In directory data blocks, the following information is checked for inconsistencies:

- Non-assigned directory

  If the I number in a directory data block refers to a non-assigned Inode, *fsck* removes this entry.
- Incorrect I number

  If the I number refers to a directory entry that is not from the Inode list, *fsck* removes this directory entry. This situation arises if incorrect data is entered in a directory data block.
- Invalid "." and ".□." entries

  The I number entry for the directory "." must be the first entry in a directory data block. The I number for "." must refer to itself, i. e. it must be the same as the I number for the directory data block. The I number entry for the directory ".□." must be the second entry in a directory data block. Its value must be the Inode of the superordinate directory entry (or the Inode of the same directory data block, if this is the root directory). If the I numbers of the directories are invalid, *fsck* replaces them with correct values. If several fixed references refer to a directory, the first one found is regarded as the superordinate directory, to which ".□." is to refer; *fsck* recommends that any other names found should be deleted.
- Isolated directories

  fsck checks the overall structure of the file system. If directories refer to something outside the file system, *fsck* guides them back into the file system by saving them in the *lost+found* directory.

7.2.4 Executing fsck for a ufs file system

This section explains each error message, the possible responses, and the associated error states. The error descriptions are assigned to the particular phase of the *fsck* program where they might occur. Faults that can occur in more than one phase are handled during initialization.

7.2.4.1 The initialization phase

Before a consistency check can be performed on a file system, certain tables must be created and certain files must be opened. The messages in this section refer to faults which stem from command line options, storage requirements, the opening of files, the status of files, file system size checks, and the creation of the work file.

cannot alloc *NNN* bytes for blockmap
cannot alloc *NNN* bytes for freemap
cannot alloc *NNN* bytes for statemap
cannot alloc *NNN* bytes for lncntp

A request made by the command *fsck* for more storage space for its virtual storage tables cannot be fulfilled. If this happens, *fsck* is terminated. This is a serious system error, which must be dealt with immediately. You should contact the Service department.

Can't open checklist file: *F*

The checklist of the file system or the file with the default values *F* (normally */etc/vfstab*) cannot be opened for reading. If this happens, *fsck* is terminated. Check the access rights for *F*.

Can't stat root

A request from *fsck* for statistics on the root directory could not be fulfilled. If this happens, *fsck* is terminated. You should contact the Service department.

Can't stat *F*
Can't make sense out of name *F*

A request from *fsck* for statistics on the *F* file system could not be fulfilled. In interactive mode, *fsck* ignores this file system and then checks the next file system specified. Check the access rights for *F*.

Can't open *F*

fsck cannot open the *F* file system. In interactive mode, *fsck* ignores this file system and then checks the next file system specified. Check the access rights for *F*.

F: (NO WRITE)

Either the *-n* option was specified or the attempt made by *fsck*, to open and read the file system *F* has failed. When *fsck* is in interactive mode, all diagnostic messages are output, but *fsck* does not attempt to make corrections.

file is not a block or character device; OK

The user inadvertently gave *fsck* the name of a regular file. Check the type of the specified file. Possible responses to the *OK* prompt are:

YES
: Ignore the error.

NO
: Ignore this file system and check the next file system.

UNDEFINED OPTIMIZATION IN SUPERBLOCK (SET TO DEFAULT)

The optimization parameter for the superblock is either *OPT_TIME* or *OPT_SPACE*. Possible responses to the *SET TO DEFAULT* prompt are:

YES
: Make a request in the superblock for optimization, so that system runtime is minimized. *tunefs* can be set to minimize disk space requirement.

NO
: Ignore the error.

IMPOSSIBLE MINFREE=D IN SUPERBLOCK (SET TO DEFAULT)

The reserve space in the superblock has increased to more than 99 percent or decreased to less than 0 percent. Possible responses to the *SET TO DEFAULT* prompt are:

YES
: Set the *minfree* parameter to 10 percent. When using *tunefs*, any percentage can be used.

NO
: Ignore the error.

MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
*N*CYL DOES NOT JIVE WITH *NCG***CPG*
SIZE PREPOSTEROUSLY LARGE
TRASHED VALUES IN SUPER BLOCK
F: BAD SUPER BLOCK: *B*
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE
SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(1M).

The superblock has been corrupted. Select an alternative superblock by calculating the displacement address, or contact the Service department. A good starting point is to select block 32.

INTERNAL INCONSISTENCY: *M*

An unrecoverable error with the message *M* has occurred in *fsck*. You should contact the Service department.

CAN NOT SEEK: BLK *B* (CONTINUE)

The request from *fsck*, to address the block number *B* in the file system has failed. You should contact the Service department. Possible responses to the *CONTINUE* prompt are:

YES

An attempt is made to continue checking the file system. This error prevents the complete checking of the file system. To check the file system once more, *fsck* should run once more. If the block was in the cache buffer of the virtual memory, *fsck* terminates with the following message:

Fatal I/O error

NO

Terminate the program.

CAN NOT READ: BLK *B* (CONTINUE)

The request from *fsck*, to read the block number *B* in the file system has failed. You should contact the Service department. Possible responses to the *CONTINUE* prompt are:

YES

fsck makes a second read attempts and outputs the following message:

THE FOLLOWING SECTORS COULD NOT BE READ: *N*

N specifies the sectors that could not be read. Each attempt that *fsck*, makes to write to a block with read errors results in the following message:

WRITING ZERO'ED BLOCK *N* TO DISK

N specifies the sector filled with binary zero. If there are hardware problems on the disk, this problem will persist. This error prevents the file system from being checked completely. To check the file system once more, *fsck* should run once more. If the block was contained in the cache buffer of the virtual memory, *fsck* terminates with the following message:

Fatal I/O error

NO

Terminate the program.

CAN NOT WRITE: BLK *B* (CONTINUE)

A request made by *fsck*, to write the block number *B* has failed. Check the disk write protection. If this is not the cause of the problem, you should contact the Service department. Possible responses to the *CONTINUE* prompt are:

YES

An attempt is made to repeat the write operation. The sectors that could not be written are displayed with the following message:

THE FOLLOWING SECTORS COULD NOT BE WRITTEN: *N*

N specifies the sectors that could not be written. If this is due to hardware problems on the disk, this problem will persist. This error prevents the file system from being checked completely. To check the file system again, *fsck* should run once more. If the block was in the cache buffer of the virtual memory, *fsck* terminates with the following message:

Fatal I/O error

NO

Terminate the program.

bad inode number *DDD* to ginode

An internal error was caused by an attempt to read an unavailable Inode *DDD*. This error terminates *fsck*. If this happens, you should contact the Service department.

7.2.4.2 Phase 1: Checking blocks and sizes

This phase checks the Inode list. It reports errors that occur when the following actions are being performed:

- Checking the Inode type
- Creating the null reference table
- Checking the Inode block numbers for faulty or duplicate blocks
- Checking the Inode size
- Checking the Inode format

Except for *INCORRECT BLOCK COUNT* and *PARTIALLY TRUNCATED INODE*, all errors in this phase result in termination of the program if the file system is being cleaned up at the time.

UNKNOWN FILE TYPE I=*I* (CLEAR)

The mode entry of the Inode *I* indicates that the file for this Inode is neither a character nor a block-oriented file, nor a socket file, normal file, symbolic reference, FIFO file, or directory. Possible responses to the *CLEAR* prompt are:

YES

Release the Inode *I* by filling with binary zero. In phase 2, this procedure always generates an *UNALLOCATED* error message for each directory entry which points to this Inode.

NO

Ignore the error.

PARTIALLY TRUNCATED INODE I=*I* (SALVAGE)

fsck found an Inode *I*, whose specified file size is less than the number of reserved data blocks. This error really only occurs if the system crashed while a file was being truncated. When cleaning up the file system, *fsck* truncates the file to the specified size. Possible responses to the *SALVAGE* prompt are:

YES

The file is truncated to the size specified in the Inode.

NO

Ignore the error.

LINK COUNT TABLE OVERFLOW (CONTINUE)

An internal table for *fsck*, which contains reserved Inodes with a zero reference counter, has run out of space. Possible responses to the *CONTINUE* prompt are:

YES

Continue the program run. This error prevents the file system from being checked completely. To check this file system again, *fsck* should run once more. If another reserved Inode with a zero reference counter is found, the error message is repeated.

NO

Terminate the program.

B BAD I=*I*

Inode *I* contains a block number *B*, which is either less than the number of the first data block, or greater than the number of the last block in the file system. This error can result in the error message *EXCESSIVE BAD BLKS* in phase 1, if the Inode *I* refers to too many block numbers outside the file system area. In phases 2 and 4, this error results in the error message *BAD/DUP*.

EXCESSIVE BAD BLKS I=*I* (CONTINUE)

The file system associated with Inode *I* contains too many blocks (generally more than 10), whose numbers are either less than the number of the first data block or greater than the number of the last block in the file system. Possible responses to the *CONTINUE* prompt are:

YES

The remaining blocks in this Inode are ignored and the next Inode in the file system is checked. This error prevents the file system from being checked completely. To check the file system again, *fsck* should run once more.

NO

Terminate the program.

BAD STATE *DDD* TO BLKERR

An internal error destroyed the *fsck* state table and assigned it the illegal value *DDD*. *fsck* is terminated immediately. If this happens, you should contact the Service department.

B DUP I=*I*

Inode *I* contains a block number *B* that has already been assigned to another Inode. This error can result in the error message *EXCESSIVE DUP BLKS* in phase 1, if Inode *I* contains too many blocks that are already assigned to other Inodes. This error calls phase 1B and creates the error message *BAD/DUP* in phases 2 and 4.

BAD MODE: MAKE IT A FILE?

This message is created if the status entry for a specified Inode is set completely at 1, in order to indicate an error in the file system. This message only indicates a disk error if it appears again after *fsck -y* has run. Here, *y* ensures that *fsck* reinitializes the Inode with a suitable value.

EXCESSIVE DUP BLKS I=*I* (CONTINUE)

Too many blocks (generally more than 10) have already been assigned to other Inodes. Possible responses to the *CONTINUE* prompt are:

YES

The remaining blocks of this Inode are ignored and checking begins with the next Inode in the file system. This error prevents the file system from being checked completely. *fsck* needs to be started once more to check the file system again.

NO

Terminate the program.

DUP TABLE OVERFLOW (CONTINUE)

An internal table in *fsck*, which contains duplicate block numbers has no more space. Possible responses to the *CONTINUE* prompt are:

YES

Continue program. This error prevents the file system from being checked completely. fsck needs to be started once more to check the system again. If another duplicate block is found, the error message is repeated.

NO

Terminate the program.

PARTIALLY ALLOCATED INODE I=*I* (CLEAR)

Inode *I* is neither reserved nor released. Possible responses to the *CLEAR* prompt are:

YES

Inode *I* is released by filling with binary zero.

NO

Ignore the error.

INCORRECT BLOCK COUNT I=*I* (*X* should be *Y*) (CORRECT)

The block count in Inode *I* is *X*, but it should be *Y*. The number is corrected when the file system is cleaned up. Possible responses to the *CORRECT* prompt are:

YES

Set the number of blocks in Inode *I* to *Y*.

NO

Ignore the error.

7.2.4.3 Phase 1B: Repeated search of duplicate blocks

If a duplicate block is found in the file system, the file system is searched again to find the Inode to which the block was originally assigned. The following information appears if the duplicate block is found:

B DUP I=*I*

Inode *I* contains the block number *B*, which has already been assigned to another Inode. This error creates the message *BAD/DUP* in phase 2. By checking this error condition, and the *DUP* error condition in phase 1, it is possible to ascertain which Inodes are using duplicate blocks.

7.2.4.4 Phase 2: Checking pathnames

This phase deletes any directory entries found in phases 1 and 1B which refer to faulty Inodes. The following errors appear here:

- Invalid root mode and Inode status
- Invalid Inode references in the directory
- Directory entries which refer to faulty Inodes
- Violations of directory integrity

All the errors that occur in this phase cause the program to crash if the file system is being cleaned up at the time. Exceptions to this rule are irrelevant fixed references, and directories whose size is not a multiple of the block size.

ROOT INODE UNALLOCATED (ALLOCATE)

The root Inode (normally Inode number 2) does not have any allocation bits. Possible responses to the *ALLOCATE* prompt are:

YES

Define Inode 2 as a root Inode. The files and directories that are generally found in root are restored in phase 3 and saved to the *lost+found* directory. If the attempt to reserve the root Inode fails, *fsck* terminates with the following message:

CANNOT ALLOCATE ROOT INODE

NO

Terminate the program.

ROOT INODE NOT DIRECTORY (REALLOCATE)

The root Inode (normally Inode number 2) of the file system is not a directory Inode. Possible responses to the *REALLOCATE* prompt are:

YES

Delete contents of the root Inode and reserve the Inode again. The files and directories that are normally in the root are restored in phase 3 and saved to the *lost+found* directory. It the attempt to reserve the root Inode fails, *fsck* terminates with the following message:

CANNOT ALLOCATE ROOT INODE

NO

fsck issues the prompt *FIX*. Possible responses to the *FIX* prompt are:

YES

The type of the root Inode is set to directory. If the root Inode data blocks are not directory blocks, a large number of error messages are output.

NO

Terminate the program.

DUPS/BAD IN ROOT INODE (REALLOCATE)

Phase 1 or phase 1B found duplicate or faulty blocks in the root Inode (normally Inode 2) of the file system. Possible responses to the *REALLOCATE* prompt are:

YES

Delete contents of the root Inode and reserve the Inode again. The files and directories that are normally in root are restored in phase 3 and saved to the *lost+found* directory. If the attempt to reserve the root Inode fails, *fsck* terminates with the following message:

CANNOT ALLOCATE ROOT INODE

NO

fsck issues the prompt *CONTINUE*. Possible responses to the *CONTINUE* prompt are:

YES

Ignore the error *DUPS/BAD* in the root Inode and attempt to resume checking the file system. If the root Inode is incorrect, a large number of error messages may be output.

NO

Terminate the program.

NAME TOO LONG *F*

A pathname is found that is too long. This points to loops in the name area of file systems, and can occur if cyclical reference chains have been created for directories. These references must be deleted.

I OUT OF RANGE I=*I* NAME=*F* (REMOVE)

A directory entry *F* has an I number *I* that is higher than the last number in the Inode list. Possible responses to the *REMOVE* prompt are:

YES

Delete directory entry *F*.

NO

Ignore the error.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* TYPE=*F* (REMOVE)

A directory or file entry *F* refers to a non-reserved Inode *I*. The owner, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. Possible responses to the *REMOVE* prompt are:

YES

Delete directory entry *F*.

NO

Ignore the error.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* TYPE=*F* (REMOVE)

Phase 1 or phase 1B found duplicate or faulty blocks which are associated with a directory or file entry *F*, Inode *I*. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. Possible responses to the *REMOVE* prompt are:

YES

Delete directory entry *F*.

NO

Ignore the error.

ZERO LENGTH DIRECTORY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)

A directory entry *F* is size *S*, i. e. zero. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. Possible responses to the *REMOVE* prompt are:

YES

Delete the directory entry *F*; this generates the error message *BAD/DUP* in phase 4.

NO

Ignore the error.

DIRECTORY TOO SHORT I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *F* directory, whose size *S* is less than the minimum size for a directory, was found. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T* and the name *F* are output. Possible responses to the *FIX* prompt are:

YES

Increase directory size to the minimum size specified.

NO

Ignore the directory.

DIRECTORY *F* LENGTH *S* NOT MULTIPLE OF *B* (ADJUST)

A directory *F*, whose size is not a multiple of the *S* directory block size *B*, is found. Possible responses to the *ADJUST* prompt are:

YES
: Modify the directory size to suit the specified size. If the file system is being cleaned at the time, a warning simply appears and the directory is adapted.

NO
: Ignore the error.

DIRECTORY CORRUPTED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (SALVAGE)

A directory with an inconsistent internal status is found. Possible responses to the *SALVAGE* prompt are:

YES
: Delete all entries up to the next directory limit (normally a 512 byte limit). This drastic action can delete up to 42 entries; it should only be used if other attempts to restore the directory have failed.

NO
: Jump to the next directory limit and resume reading or modifying the faulty directory.

BAD INODE NUMBER FOR '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *I* directory is found, whose I number does not correspond to *I* for ".". Possible responses to the *FIX* prompt are:

YES
: Change the I number for "." so that is corresponds to *I*.

NO
: Do not change the I number for ".".

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *I* directory was found, whose first entry indicates that there is no data block available. Possible responses to the *FIX* prompt are:

YES
: Enter I number from *I* for ".".

NO
: Leave the directory unchanged.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, FIRST ENTRY IN DIRECTORY CONTAINS *F*

An *I* directory was found, whose first entry is *F*. *fsck* cannot solve this problem. The file system must be mounted and the entry *F* must be transferred to another location. The file system must then be mounted and *fsck* must be restarted.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, INSUFFICIENT SPACE TO ADD '.'

A directory *I* was found whose first entry is not ".". *fsck* cannot solve this problem. You should contact the Service department.

EXTRA '.' ENTRY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *I* directory was found that has more than one entry for ".". Possible responses to the *FIX* prompt are:

YES
: Delete the superfluous entries for ".".

NO
: Leave the directory unchanged.

BAD INODE NUMBER FOR '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *I* directory was found whose I number for ".□." does not correspond to the number for the superordinate *I* directory. Possible responses to the *FIX* prompt are:

YES

The I number for ".□." is changed to correspond to the superordinate *I* directory. Note: In the root directory, the Inode ".□." refers to itself.

NO

Leave the I number for ".□." unchanged.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *I* directory was found whose second entry is not reserved. Possible responses to the *FIX* prompt are:

YES

Enter the I number of the superordinate *I* directory for ".□.". The root Inode ".□." refers to itself.

NO

Leave the directory unchanged.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, SECOND ENTRY IN DIRECTORY CONTAINS *F*

An *I* directory was found, whose second entry is *F*. *fsck* cannot solve this problem. The file system should be mounted and the *F* entry transferred to another location. The file system should then be unmounted and *fsck* restarted.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, INSUFFICIENT SPACE TO ADD '..'

An *I* directory was found, whose second entry does not correspond to ".□." (the superordinate directory). *fsck* cannot solve this problem. The file system should be mounted and the entry *F* should be transferred to another location. The file system should then be unmounted and *fsck* restarted.

EXTRA '..' ENTRY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)

An *I* directory was found that has more than one entry for ".□." (the superordinate directory. Possible responses to the *FIX* prompt are:

YES

Delete the superfluous entries for ".□.".

NO

Leave the directory unchanged.

N IS AN EXTRANEOUS HARD LINK TO A DIRECTORY *D* (REMOVE)

fsck has found a fixed reference *N* to a directory *D*. During the cleaning process, the irrelevant references are ignored. Possible responses to the *REMOVE* prompt are:

YES

Delete the irrelevant *N* reference.

NO

Ignore the error.

BAD INODE *S* TO DESCEND

An internal error caused an *S* state, which is unusual. This was sent to the routine that searches through the directory structure of the file system in descending order. *fsck* is terminated. You should contact the Service department.

BAD RETURN STATE *S* FROM DESCEND

An internal error caused an *S* state, which is unusual. This error was returned from the routine which searches through the directory structure of the file system in descending order. *fsck* is terminated. You should contact the Service department.

BAD STATE *S* FOR ROOT INODE

An internal error caused an *S* state, which should be assigned to the root Inode. This state does not exist. *fsck* is terminated. You should contact the Service department.

7.2.4.5 Phase 3: Checking concatenation

This phase checks the directories that were searched in phase 2. It reports errors, which are the result of

- Directories for which there are no references
- *lost+found* directories that are not available, or which are full

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)

The directory Inode *I* did not refer to any directory entry when the file system was running. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. When cleaning, the directory is added again provided its size is not zero; otherwise it is deleted. Possible responses to the *RECONNECT* prompt are:

YES

Enter directory Inode *I* in the directory for lost files (generally the *lost+found* directory). This may result in a *lost+found* error message in phase 3, if there are problems entering the directory Inode *I* in the *lost+found* directory. Alternatively, it may result in a *CONNECTED* error message in phase 3 if the entry could be executed successfully.

NO

Ignore the message. This results in the error message *UNREF* in phase 4.

NO lost+found DIRECTORY (CREATE)

There is no *lost+found* directory in the root directory of the file system. During the cleanup, *fsck* attempts to create a *lost+found* directory. Possible responses to the *CREATE* prompt are:

YES

Create a *lost+found* directory in the root directory of the file system. The following message can appear here:

NO SPACE LEFT IN / (EXPAND)

If a *lost+found* directory cannot be created, the following message is displayed:

SORRY. CANNOT CREATE lost+found DIRECTORY

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

NO

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

lost+found IS NOT A DIRECTORY (REALLOCATE)

The entry for *lost+found* is not a directory. Possible responses to the *REALLOCATE* prompt are:

YES

Reserve the directory Inode and change *lost+found* so that the reference is created. The Inode that was previously assigned to the *lost+found* directory is not deleted. Instead, it is either reassigned as *UNREF* Inode, or its reference counter is adjusted during the continuing course of this phase (*ADJUST*). If the *lost+found* directory is not created, the following message appears:

SORRY. CANNOT CREATE lost+found DIRECTORY

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

NO

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

NO SPACE LEFT IN /lost+found (EXPAND)

There is no more space for another entry in the *lost+found* directory in the root directory of the file system. When the file system is cleaned up, the *lost+found* directory is expanded. Possible responses to the *EXPAND* prompt are:

YES

Expand the *lost+found* directory to create space for a new entry. If this fails, *fsck* outputs the following message:

SORRY. NO SPACE IN lost+found DIRECTORY

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4. Delete unnecessary entries from the *lost+found* directory. This error aborts the program if the system is being cleaned at the time.

NO

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

DIR I=*I1* CONNECTED. PARENT WAS I=*I2*

This message is purely informative. It indicates that the directory Inode *I1* has been linked successfully to the *lost+found* directory. Inode *I2*, which is superordinate to the directory Inode *I1* was replaced by the I number of the *lost+found* directory.

DIRECTORY *F* LENGTH *S* NOT MULTIPLE OF *B* (ADJUST)

An *F* directory is found whose size *S* is not a multiple of the block size for *B* directories. (This can happen again in phase 3, if the error is not corrected in phase 2.) Possible responses to the *ADJUST* prompt are:

YES

Adapt the length of the particular block size. If the file system is being cleaned at this time, a warning simply appears and the directory is adapted.

NO

Ignore the error.

BAD INODE *S* TO DESCEND

An internal error created an *S* state, which is very unusual. This was transferred to the routine, which searches the directory structure of the file system in descending order. *fsck* is terminated. You should contact the Service department.

7.2.4.6 Phase 4: Checking the reference counter

This phase checks the information that the reference counter compiled in phases 2 and 3. It reports errors, which are the result of

- Directories for which there are no references
- A *lost+found* directory that is not available or is full
- Incorrect reference counters for files, directories, symbolic references, or special files
- Files, symbolic references and directories for which there is no reference
- Faulty or duplicate blocks in files, symbolic references and directories

All the errors in this phase can be corrected when the system is being cleaned (unless there is no more space in the *lost+found* directory).

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)

The directory Inode *I* was not linked to a directory entry while the file system was running. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. The directory is deleted when the system is being cleaned if its size or reference counter is exactly zero; otherwise it is relinked. Possible responses to the *RECONNECT* prompt are:

YES

Relink Inode *I* to the file system's directory for lost files (normally *lost+found*). This can result in the error message *lost+found* in phase 4, if there are problems with the link between Inode *I* and the *lost+found* directory.

NO

Ignore the error. This results in the error message *CLEAR* in phase 4.

(CLEAR)

The Inode mentioned in the last error message cannot be relinked. This message cannot appear when the file system is being cleaned, because lack of space when relinking a file is an error which causes the system to crash. Possible responses to the *CLEAR* prompt are:

YES

Release the Inode by filling with binary zero.

NO

Ignore the error.

NO lost+found DIRECTORY (CREATE)

There is no *lost+found* directory in the root directory of the file system. When cleaning up the system, *fsck* tries to create a *lost+found* directory. Possible responses to the *CREATE* prompt are:

YES

Create a *lost+found* directory in the root directory of the file system. This can result in the following message:

NO SPACE LEFT IN / (EXPAND)

If a lost+found directory cannot be created, the following message is output:

SORRY. CANNOT CREATE lost+found DIRECTORY

The attempt to relink a lost Inode failed. This results in the error message *UNREF* in phase 4.

NO

The attempt to relink the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

lost+found IS NOT A DIRECTORY (REALLOCATE)

This entry for *lost+found* is not a directory. Possible responses to the *REALLOCATE* prompt are:

YES

Reserve directory Inode and modify the *lost+found* directory so that the reference is created. The Inode, which was previously assigned to the lost+found directory is not deleted. Instead, it is either reassigned as an *UNREF* Inode, or its reference counter is adjusted during the continuing course of this phase (*ADJUST*). If the *lost+found* directory cannot be created, the following message appears:

SORRY. CANNOT CREATE lost+found DIRECTORY

The attempt to link the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

NO

The attempt to link the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

NO SPACE LEFT IN /lost+found (EXPAND)

There is no space for another entry in the *lost+found* directory in the root directory of the file system. When the system is cleaned up, the *lost+found* directory is expanded. Possible responses to the *EXPAND* prompt are:

YES

Expand the *lost+found* directory to create space for a new entry. If this fails, *fsck* outputs the following message:

SORRY. NO SPACE IN lost+found DIRECTORY

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4. Delete unnecessary entries from the *lost+found* directory. This aborts the program, if the system is being cleaned at the time.

NO

The attempt to incorporate the lost Inode is aborted. This results in the error message *UNREF* in phase 4.

LINK COUNT TYPE I=I OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)

The reference counter for Inode *I* is *X*, but it should be *Y*. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. When the system is being cleaned the reference counter is adapted, unless the number of references increases. Normally, this would never happen, unless there is a hardware error. If the number of references increases during the cleaning process, *fsck* is terminated with the following message:

LINK COUNT INCREASING

Possible responses to the *ADJUST* prompt are:

YES
: Replace the file Inode *I* with *Y*.

NO
: Ignore the error.

UNREF TYPE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

The directory Inode *I* was not connected with a directory entry while the file system was running. The owner *O*, the mode *M*, the size *S*, the time of the last modification *T*, and the name *F* are output. As this file was not connected because its size or reference counter was zero, it is deleted when the system is being cleaned up. Possible responses to the prompt *CLEAR* are:

YES
: Delete Inode *I* by filling it with binary zero.

NO
: Ignore the error.

BAD/DUP TYPE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Phase 1 or phase 1B found duplicate or faulty blocks, which were connected with the Inode *I*. The owner *O*, the mode *M*, the size *S*, the time of the last change *T*, and the name *F* are output. This message cannot appear when the file system is being cleaned, because it would have caused a program crash earlier. Possible responses to the *CLEAR* prompt are:

YES
: Release Inode *I* by filling it with binary zero.

NO
: Ignore the error.

7.2.4.7 Phase 5: Checking cylinder groups

This phase checks free blocks, as well as the Inode mappings used. Errors are reported, which are the result of

- reserved blocks that appear in lists of free blocks
- free blocks that do not appear in lists of free blocks
- incorrect total number of free blocks
- free Inodes that appear in lists of reserved nodes
- reserved Inodes that do not appear in the lists of Inodes currently being used
- incorrect total number of Inodes currently being used

CG *C*: BAD MAGIC NUMBER

The magic number of the cylinder group *C* is invalid. This indicates that the mappings of cylinder groups were destroyed. In interactive mode, the cylinder group is given a mark which indicates that it must be reconstructed. If this error occurs when the file system is being cleaned, it aborts the program.

BLK(S) MISSING IN BIT MAPS (SALVAGE)

Some free blocks are missing in the block list for a cylinder group. The lists are reconstructed when the file system is being cleaned. Possible responses to the *SALVAGE* prompt are:

YES
: he list of free blocks.

NO
: Ignore the error.

SUMMARY INFORMATION BAD (SALVAGE)

Errors were discovered during the end evaluation. The end evaluation is performed again when the file system is cleaned. Possible responses to the *SALVAGE* prompt are:

YES
: Reconstruction of the end evaluation.

NO
: Ignore the error.

FREE BLK COUNT(S) WRONG IN SUPERBLOCK (SALVAGE)

Incorrect information about free blocks was found in the superblock. This information is recompiled if the file system is being cleaned. Possible responses to the *SALVAGE* prompt are:

YES
: Reconstruction of the information about the free blocks in the superblock.

NO
: Ignore the error.

7.2.4.8 Cleanup phase

Once a system has been checked, certain cleaning functions are executed. These output messages about the state of the file system.

V files, *W* used, *X* free (*Y* frags, *Z* blocks)

This message indicates that the checked file system contains *V* files that are using a *W* fragment block size, and that *X* fragment-size blocks are free in the file system. The numbers in brackets split the free number into *Y* free fragments and *Z* free, fullsize blocks.

***** FILE SYSTEM WAS MODIFIED *****

This message indicates that *fsck* modified the file system. If the file system is mounted, or if it is the current file system, it has to be reloaded. If the file system is mounted, it is important that you unmount it and restart *fsck*, to avoid the risk of undoing the work done by *fsck* by copying the tables into the storage area.

Glossary

boot block, load block

This area contains initial program loader programs which start the UNIX system.

cylinder group block

A cylinder group contains several Inodes specified in the Illist for the cylinder group, as well as a variety of relevant data blocks. The size of the cylinder group block depends on the size of the file system.

data blocks

The largest area of the file system is devoted to the data blocks used by the files. In the case of the ufs file system, it was attempted to place the data blocks in cylinder groups. The physical size of each block depends on how the file system has been formatted with mkfs(1M). Depending on the file system type, it can be formatted in 512, 1024, 2048, 4096 or 8192 byte blocks.

Illist

The maximum number of files that can be created in a file system depends on the number of available Inodes. Consequently, if a system has 100 MB of unused storage space but only one available Inode, only one further file can be created regardless of the size of the file.

Inode

An Inode (Index Node, Interior Node, Information Node) describes a file and its owner and also contains information on the position of the file's data blocks on the storage medium.

superblock

The superblock is used exclusively as an area for managing the entire file system. It contains a list of the free blocks and free Inodes which are assigned to files as well as other administrative information. A ufs file system contains several copies of the superblock in order to guarantee the integrity of the file system. Each copy of the superblock is 192 bytes large; only the superblock is used when installing the file system. The critical information in a block is static, i.e. does not change while the file system is being used. A copy of the superblock can be used as a backup of the primary superblock was corrupted or otherwise became inconsistent when the system was booted.

Abbreviations

CD-ROM

Compact Disc - Read Only Memory

HS

High Sierra/ISO9660 File System

NFS

Network File System

UFS

UNIX File System

VxFS

VERITAS File System

Related publications

Ordering manuals

The manuals listed above can be ordered from your local Siemens Nixdorf office.

[1] **SINIX V5.4**
System Administrator's Guide

Target Group
System Administrators

Contents
Introduction to the system administration of SINIX systems

[2] **Reliant UNIXV 5.44**
System Administrator's Reference Manual

Target Group
System administrators

Contents
Description of commands, application programs, file formats, and special files for system administration

[3] **Reliant UNIX V5.44**
Commands

Target Group
System administrators, Users

Contents
Description of the user commands of the Reliant UNIX operating system

[4] **Reliant UNIX V5.44**
Network Administrator Guide

Target Group
System administrators

Contents
Description of the basic functions of the network administration

[5] **Reliant UNIX 5.44**
VERITAS File System (VxFS) - System Administrator Guide

Target Group
System administrators

Contents
Description of the VERITAS file system

[6] **SINIX V5.41**
CD ROM - Notes for System Administrators

Target Group
System administrators

Contents
Description of the High-Sierra/ISO9660 file system