NAME

check − file system consistency check and repair

SYNOPSIS

check [**−syna**] [filesystem] ...

DESCRIPTION

Check audits UNIX file systems for consistency and corrects any discrepancies. Since these corrections will, in general, result in a loss of data, the program will request operator concurence for each such action. All questions should be answered by typing "**yes**" or "**no**", followed by a line feed. Typing "**yes**" will cause the correction to take place. However, if the program does not have write permission on the file system or the no option, **−n**, is on, then all questions will automatically be answered "**no**". Alternatively the yes option, **−y**, will cause all questions to be answered "**yes**". If the **-a** option is supplied then check will attempt an "automatic" check. In this case it will answer "yes" only to correct nonsevere errors. In all other cases check will loop and print a message requesting manual intervention.

The program consists of six separate phases. Some phases are skipped if they are not needed. In phase one, *check* examines all block pointers in all files, checking for pointers which are outside of the file system (BAD) and for blocks which appear in more than one file (DUP). A table is made of all DUP blocks and all defective files are marked for clearing. Each error is printed, but no correction takes place in this phase.

The second phase is run only if DUP blocks were found in phase one. This phase finds the rest of the DUP blocks; marking each for clearing.

The third phase checks the directory structure of the file system. This is done by descending the directory tree, examining each entry. A count is kept of the number of references to each file. If any entry refers to an unallocated file, a file marked for clearing, or a file number outside the file system; then the entry is printed and, if the operator agrees, it is removed. Refusing to remove an entry to a marked file will clear the mark, preserving the file and its subsequent entries.

In phase four all marked or unreferenced files are listed. With concurence from the operator, each of these files is then cleared. In addition, any file whose link count does not agree with the number of references is listed, and, if agreed, the link count is adjusted.

If the salvage option, −s, is on, then phase five is skipped. Otherwise, *check* examines the free list. If any blocks are found which are outside the file system or which have been previously encountered in a file or elsewhere in the free list, then the list is pronounced BAD and a salvage is called for. Agreement will set the salvage option and proceed to the next phase. If there are no defects in the free list and all blocks are accounted for, the check is finished. Otherwise, the number of missing blocks (i.e. in neither a file nor the free list) is printed and a salvage is requested.

The last phase is the salvage operation, where the free list is recreated. It is run whenever the salvage option is on or a problem has been found with the free list. Simply stated, a new free list is constructed containing all blocks not found in some file.

The system responses are in general self explanatory and follow the sequence described above. In the specification that follows, the following notation will be used:

<b>	block number
<i>	inode number
<fname>	file pathname
<n>	positive integer

<c> option character

Check begins with the following output:

```
            <filesystem>{(NO WRITE)}
Phase 1 − Check Blocks
```

The "(NO WRITE)" message indicates that the program does not have write permission on the file system. Therefore, subsequent corrections will be suppressed by automatically answering "**no**" to all questions. Phase one then proceeds to list any BAD or DUP blocks and their inode number, as follows:

```
    <b>  BAD   I = <i>
    <b>  DUP   I = <i>
    <b>  EXCESSIVE DUPS   I = <i>
```

If too many DUPs are encountered, the program will list all blocks, but will not mark the excess DUPs for later processing. When Phase 1 is finished, Phase 2 is run if any DUPs were encountered. Otherwise, Phase 2 is skipped. This Phase will list the rest of the DUP blocks as follows:

```
Phase 2 − Rescan for more DUPS
    <b>  DUP   I = <i>
```

Check now descends the directory tree, asking to remove any defective entries.

```
Phase 3 − Check Pathnames
    I OUT OF RANGE  I = <i>  <fname>  REMOVE?
    UNALLOCATED     I = <i>  <fname>  REMOVE?
    BAD/DUP         I = <i>  <fname>  REMOVE?
```

Unless the no option is on, the program will wait for a response of "**yes**" or "**no**" after each question. Note, a **no** answer to the BAD/DUP entry will unmark that inode for clearing. This will suppress any subsequent correction to that file.

Now *check* will clear or adjust any defective files. Again, it will wait for a "**yes**" or "**no**" response to each question. The program will also indicate whether each entry is a file or a directory.

```
Phase 4 − Check Reference Counts
    UNREFERENCED {FILE/DIRECTORY}      I = <i>      CLEAR?
    BAD/DUP      {FILE/DIRECTORY}  I = <i>      CLEAR?
    LINK COUNT   {FILE/DIRECTORY}  I = <i>      ADJUST?
```

If the salvage option is not on, the program will now validate the free list. Otherwise, this phase is skipped. If there are any errors in the free list, it will specify them and request a salvage.

```
Phase 5 − Check Free List
    BAD FREE LIST      SALVAGE?
    <n> MISSING        SALVAGE?
```

Phase 6 is the salvage operation. It is only done if one has been requested.

```
Phase 6 − Salvage Free List
```

Finally, some totals are printed: the total number of allocated files (including directories and special files); the number of blocks in use; and the number of blocks in the free list.

```
    <n> FILES    <n> BLOCKS <n> FREE
```

If the filesystem has been modified, then the following message is printed and the program goes into a loop. This is only a reminder to the operator since the program can be forced to terminate with a <DEL> character.

```
*****BOOT UNIX(NO SYNC!)*****
```

FILES

/dev/rootdev default file system to be checked

SEE ALSO

sync(1M), dcheck(1M), update(1M), clri(1M), crash(6), updfs(1M), fs(5),

DIAGNOSTICS

While running, a number of errors can occur which cause the *check* program to terminate. An illegal option or the inability to open the file system are shown as:

```
<c> OPTION??
CAN NOT OPEN <filesystem>
```

An I/O error on the filesystem will also cause an error message. In this case, the operator is given the choice of exiting ("**yes**") or continuing ("**no**"). This error is generally a hardware error, and continuing is rarely a good idea.

```
CAN NOT READ <filesystem>     BLOCK <b>  EXIT?
CAN NOT SEEK <filesystem>     BLOCK <b>  EXIT?
CAN NOT WRITE <filesystem>    BLOCK <b>  EXIT?
```