1616: User Disk

Version 4.088

August 1993

Beginner's Disk Reference and Tutorial Manual
Applix 1616 microcomputer project
Applix pty limited

1616 User Disk Reference and Tutorial Manual

Even though Applix has tested the software and reviewed the documentation, Applix makes no warranty or representation, either express or implied, with respect to software, its quality, performance, merchantability, or fitness for a particular purpose. As a result this software is sold "as is," and you the purchaser are assuming the entire risk as to its quality and performance.

In no event will Applix be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or its documentation.

The original version of this manual was written by Andrew Morton
Additional introductory and tutorial material by Eric Lindsay
Editorial and design consultant: Jean Hollis Weber

Comments about this manual or the software it describes should be sent to:

Applix Pty Limited
Lot 1, Kent Street,
Yerrinbool, 2575
N.S.W. Australia
(048) 839 372
Private Applix BBS numbers include Colin McCormack (02) 543 8213, SSM (02) 554 3114, Trantor (02) 718 6996 and PPT 544 1060. These are all "callback systems" (phone once for two rings, phone them again within 30 seconds).
User Group meetings are generally held on the second Saturday of each month, from about 1 p.m. until midnight or so. All computer enthusiasts are welcome, as are new visitors. Come in and meet the designer of the Applix system.

ISBN 0 947341 xx x

MC68000®™ is a trademark of Motorola Inc.

1
Preparing New Disks

A number of utility programs to prepare disks for use are supplied on the 1616/OS V4 User disk. This chapter details how to prepare a new disk for use, how to copy existing disks, how to check your disks for faults, and also how to automatically repair disk faults. Once you get used to the three programs described here, you should be able to use them from day to day without further reference to this chapter.

This chapter does not include any technical details of disk operation; technical details are included in the *Disk Co-processor Manual*, which also contains instructions for building your disk controller card.

The various programs described in this chapter are assumed to be in a directory named `bin` on the disk in drive `/f0`. Check that they are in their correct place by doing a `dir /f0/bin`. If they are not there, use the `copy` command (described in your original *User Manual*) to copy them into `/f0/bin`.

We also assume that your `xpath` includes `/f0/bin`. Check that this is true by typing `xpath`. If the line displayed does not include `/f0/bin`, correct your xpath by typing `xpath + /f0/bin` so that `bin` is added to your xpath.

Please note that if your disk is not set up as described above, some of these programs will not work as described. If a program fails to work as described, the first things to check are the location on disk of the program, and that your xpath is correct. The more you use your Applix 1616, the more you will appreciate the importance of correct program location on disk, and correct xpaths.

Initialising block devices

Before a brand new, never before used disk can be used, it must be formatted and initialised to suit the type of computer in which it will be used. This process need only be done once in the life of a disk. It involves putting magnetic tracks on the disk, and setting some tracks up in a particular form. **Note that this totally destroys anything previously on the disk**. You **must** make very sure that a disk does not contain anything of value before you initialise it.

The program `blockdev.xrel` permits the initialisation of block devices, formatting of floppies and the conversion of 1616/OS V2.X disks to 1616/OS V4.*n*. This is how you make yourself a totally new disk, ready for use in your 1616.

To run this program type

```
blockdev devname
```

Where `devname` is the name of the device (/f0, /f1, etc) which you wish to format, initialise, etc.

For example, if you have a single disk drive, put your User Disk in `/f0` and type `blockdev /f0`. The program will prompt you to remove your User Disk and replace it with the disk you wish to initialise.

There are three levels of disk preparation available with this `blockdev` program. The most basic level is to simply alter the disk's boot sector program; you enter the name of the new boot program and this is written onto the disk. If no boot program exists on a new disk, then the new disk is skipped in the booting sequence. If you wish to boot (start the computer) from a particular disk, it must contain a boot program. The exact details of what is in the boot program is rather involved, and is explained in the *Technical Reference Manual*, however a step by step tutorial, with some explanation, is provided below.

In the next level of disk preparation you may 'initialise' a disk. This recreates the disk's root block, bitmap and directory. All files are lost. The boot sector program in the boot block needs to be rewritten after this.

The most intensive level of disk preparation involves a physical format of the disk, followed by initialisation, followed by the boot sector program. This is what is required for a totally new disk. This will only work on the /f0 and /f1 devices, since these are the only physical devices which this program knows how to format.

Another option with this program is to upgrade a Version 2 disk to 1616/OS V4. This alters an old Version 2 disk's root block into the correct format. The V2.X directory becomes the V4.0 root directory. The disk is still readable and writeable under 1616/OS V2.X. This option may be used for altering a disk's `volume` name, as well as upgrading the version.

How do you format a floppy?

Like most operating systems, 1616/OS does not contain routines to do everything you might need. For lengthy, or infrequently used activities, like formatting, special programs are supplied on disk. You will need your 1616/OS User Disk (supplied when you bought 1616/OS version 3 or 4), which has a `/bin` directory which contains the format program `blockdev` (version 2 of 1616/OS used `sddutil`, which is now obsolete). We will assume that you only have a single disk drive; if you have two, you can put the blank disk to be formatted in drive `/f1` instead of swapping disks. Before you start, **write protect** your User disk against mistakes, by flicking open the write protect clip in the corner so you can see through the hole.

First, let me mention that there are two types of disks that can be produced. The normal one, which stores programs and data, and a boot disk. The normal disk, as produced by a simple format, is not really suitable for booting your Applix (although the Applix will start up regardless of what sort of disk you use).

The boot disk starts up your Applix in a particular way, and can customise what happens when it boots. It can do this because it contains a boot program. The normal boot program, `bootv3.exec` (sometimes named `bootv3`) supplied in your User Disk `/sys` subdirectory, only does a few very simple things. It sets the step rates for /f0 and /f1, and then checks the reset level. If you pressed the reset button, it doesn't do anything else, except return control. If it is a Level 0 (power on)

reset, it changes the default drive to whatever drive you booted from. When you are more experienced, you can write your own 'custom' boot program, to do more elaborate things.

The Applix 1616 will also search for and use a set of memory resident driver programs, which are combined in a file normally named `mrdrivers`. This installs several modifications to 1616/OS. For example, on the User Disk, the usual `mrdrivers` file will install a time of day clock (tdos.mrd), which displays the time in the top right hand corner of the display.

Next, a start up program is run. The name of this program is provided by the boot program, and it is usually named `autoexec.shell`. If you read the source code `bootv3.s` on your User Disk, you will see where the name 'autoexec' appears. If you write a custom boot program, this name can be changed. The `autoexec.shell` program can contain almost any set of commands that you like.

Now that we have sketched the background, let us go through formatting an ordinary, non-boot, disk.

Place the User Disk in drive `/f0`, and change to the **/bin** subdirectory by typing `cd bin` (if your `xpath` includes `/f0/bin` then you need not change to that directory).

Start the program by typing `blockdev /f0`. If you have two disk drives you can specify `/f1` instead.

The **blockdev** program will put a menu on your screen. Select the first option, 0. The program will prompt you to put a blank disk in the drive, and press the Spacebar to start the format.

To produce an ordinary disk, just press Enter as the answer to all questions. That is all there is to it.

The only real complication is that you can not boot your Applix using this disk. Disks that do not have a boot program are skipped when the Applix boots. In my case, this simply means the Applix will end up booting from the hard disk, with the correct `autoexec.shell` and so on. If you don't have a hard disk, the Applix will start without running the boot program, without any mrdrivers, and without running an autoexec.shell file. Things will still work, but you should have a boot disk somewhere to start the system with everything set up to suit your way of working.

You will find an example of what is contained in the boot program in the `bootv3.s` source code in the **sys** subdirectory on your User disk.

Since making a boot disk can be somewhat complicated, I'd suggest that the first time, the novice simply copy their **User** Disk (which is correctly set up as a boot disk) to a newly formatted disk using the `diskcopy` utility (described below) on the User Disk. Then just delete all the files you no longer need on the new disk.

If you wish, you can change the name of your new disk, using the **blockdev** utility (the original name you put on while formatting was overwritten by the `diskcopy`.)

If you really want to do the entire format of a bootable disk, here is a more complete description (assuming one drive).

Start with the User Disk in /f0, and start **blockdev**, by typing `blockdev /f0`.

Select option 0. Press Enter for 80 tracks, and again for 64 root directory entries.

Edit the disk name to whatever you like, say `/boot`.

Enter the boot program name as `/f0/sys/bootv3.exec` (or `/f0/sys/bootv3` if that name is used).

Put the blank disk in drive /f0 when prompted, and press the spacebar. When the format is finished, quit from **blockdev** using option 4.

Remove the newly formatted disk, and replace the User Disk in the drive. Copy **all** the files you need to the ram disk /rd. The files you need include `mrdrivers`, and `autoexec.shell` from the root directory on the Users Disk. You may also need some extra files from other directories. For example, I need Jeremy Fitzhardinge's `swget.xrel` program, which is usually in my /bin directory, and some early Applix boot disks need the `ibmfont.xrel` file which is also in the /bin subdirectory.

After copying these files to /rd, place your newly formatted disk back in the drive, and copy the files from the ram disk to /f0. You will have to also create a /bin directory, for any files that usually reside in a /bin directory (or else change the contents of the `xpath` command in the `autoexec.shell` file to coincide with where the programs now exist).

You should now have a complete bootable disk, that will start your Applix just like the original.

Copying disks

The program `diskcopy.xrel` copies a 1616/OS disk block-for-block to another disk. The copy is identical, except the DATE and SPECIAL fields in the rootblock of the target disk are altered to prevent the system from assuming the two disks are the same (a full description of these fields is in the *Disk Coprocessor Manual*). `Diskcopy` is the program you use when you need to make a copy of an entire disk as easily as possible.

Usage of this program:

```
diskcopy sourcedev destdev [-f] [-v] [-r]
```

Where `sourcedev` is the device identifier of the source disk (/f0, /f1, etc) and `destdev` is the destination disk identifier. If these are the same then you will be prompted to perform disk swaps at the appropriate times (although with one drive, this can be tiresome).

The `-f` flag forces physical formatting of the destination disk, saving you the bother of using `blockdev`. This only works if the destination device is /f0 or /f1 (that is, it won't work on hard disks or ram disk). The program will prompt you to press the Spacebar prior to doing the formatting. This means that you do not need to use `blockdev` if only copying disks.

The `-v` flag sets verbose mode: the program prints out more status information as the copy proceeds to reassure you things are working. There is no harm in using it, however later you may wish to find ways to copy disks without further intervention.

The `-r` flag suppresses the re-randomisation of the root block's DATE and SPECIAL fields, yielding an exact copy of all blocks of the disk. This is not the best of ideas under normal circumstances.

An example of the use of this program:

```
diskcopy /f0 /f1 -v -f
```

This will copy from /f0 to /f1, formatting /f1 and printing out status information as the copy proceeds.

Checking and repairing file systems

The `fscheck.xrel` utility scans the file system on a disk, reporting any inconsistencies in information on the disk. In short, it both tests and repairs damaged disks, provided the damage is not too severe. This is a very handy program. You should check any disk that was in your system during a crash, just as a precaution. Also, check any disk that appears to be giving problems.

For the first test, use the **-q** option, since it saves time.

To use this program, type

```
fscheck devname [-v] [-y] [-yy] [-q]
```

Where `devname` is /f0, /f1, etc.

The `-v` option, if specified, causes the program to operate in verbose mode, so more status information is printed.

If the `-y` option is provided this program automatically answers 'yes' to all its questions, except for the last one where it asks if you wish to write out the disk bitmap; this question must be manually answered.

If the `-yy` option is given all questions, including the last one, are automatically answered in the affirmative. Use this option for a fully automatic repair.

If the `-q` option is given, a simplified and much faster check is made. If a file system passes this without complaining, it should be OK. If it complains, then use the other options for more information.

`fscheck.xrel` does not check the disk for bad disk blocks - these may or may not be detected, depending upon where on the disk they lie.

Errors detected in operation are repaired by deleting the offending block from the file. The file length is reduced by 1024 bytes (unless the block is at the end of the file). A message is displayed and you are asked whether or not the fix is to be made. The file will probably be corrupted. If a file's block map block number is an invalid block then the file is deleted. If a directory's block range is invalid then it is removed. The user is told about all of these things and prompted to confirm. A number of these fixes will cause bitmap inconsistencies, but these are fixed later or on another pass of the file system check program.

This program should be rerun until it gives the disk a clean bill of health; it may take several passes to fully repair a disk.

If a change is made to the bitmap of a non-removable or hard disk this program exits with a ***warmboot*** system call, which is like pressing reset. This must be done because the system never re-reads the bitmap and root block of a non-removable device, so alterations to the bitmap would not be noted if this program exited in the normal manner.

Disk names

When you format a diskette using the `blockdev` utility on the user disk, you have the option of giving your diskette a meaningful name. The name can be seen when doing a directory, or by using the `volumes` command. This can be handy for keeping track of the contents of disks more easily.

2
Startup Files

The Applix 1616 will start operating without a disk in the drive, unlike many computer systems. Despite this ability, the best way to start the Applix 1616 is with a specially configured disk called a 'boot disk', described in the next section of this chapter.

The boot disk tells the 1616/OS which file to take startup commands from. This file is typically named something like `autoexec.shell`. The operation of `autoexec.shell` files is described in the second section of this chapter.

Various rather fundamental changes can be made to the operation of the 1616/OS by means of an `mrdrivers` file. This allows you to add new commands to the operating system, change existing commands, set default colours, change the ram disk size, and make other changes. Creating mrdrivers files is described in the mrdrivers section of this chapter.

Boot disk

A boot disk is one that has been produced using the `blockdev` program described in the previous Chapter. It also includes a special startup program, usually called `bootv3`, which is placed on the disk when it is first formatted by `blockdev`. This `bootv3` startup program goes in the disk bootblock, and thus is not visible in a directory.

Advanced users can specially customise their bootblock program by writing their own boot program, instead of using the sample program `bootv3.s` provided in the `/sys` directory of the *User Disk*.

The standard boot program does very little. One thing is to simply adjusts the speed of the disk drive step rate to their fastest settings. The drives are deliberately started with slow settings so that even ancient drives work correctly for boot purposes.

More important for you, the bootblock tells the 1616/OS which file contains the commands you want to run each time you start the system. These commands are held in a file named `autoexec.shell`. By altering the `bootv3` program, you can change the name of this program. You can also allow use of a different autoexec program for each reset level. This might mean that different things happen when you power up, than when you press the reset button.

Autoexec.shell files

The `autoexec.shell` file simply contains a list of commands for 1616/OS to run. All the commands in it run automatically when you power up your computer. A typical simple `autoexec.shell` file is shown below:

```
bin/swget               ; set time and date from Smartwatch
mkdir /rd/bin           ; make sure some programs can run quick
xpath bin /rd/bin       ; make all bin dirs accessible
copy bin/shutdown.xrel /rd/bin  ; copy needed programs
copy bin/startup.xrel /rd/bin
assign /hitech /f0/hitech      ; for C compiler
assign /temp /rd
assign /usr/lib /f0/usr/lib    ; set up for Dr Doc use
cd /rd                  ; go someplace quick
```

As usual with `shell` files, the **;** simply indicates a comment follows. Usually comments are not as detailed as in these examples. First, notice that the 1616/OS assumes that you are getting all your files from root directory of the boot disk. This is a consequence of the actions of the `bootv3` program held in the boot block of the boot disk.

The first command I use (`swget`) gets the date and time from a SmartWatch real time clock chip on my Applix 1616 motherboard, and tells the operating system. I always think the $50 cost of that chip and software was worthwhile, because I'm forever forgetting to use `setdate` when I start a system.

Since the ram disk is much faster than an floppy disk, it makes sense to set up a directory on it, and copy over some frequently used files. In my case, they are actually ones used for starting and stopping the hard disk motor (I obviously can't leave the startup program on the hard disk!) Floppy only users will have other commands they use frequently, perhaps diskcopy, the `ssasm` assembler, or similar. Notice that one standard place to hold executable (`.xrel`) files is in a `bin` subdirectory.

Note that the commands in the `autoexec.shell` file are simply a list of commands. They do not actually do any testing for particular conditions. We will consider ways round this lack in a later chapter.

Construct MRDRIVERS file for boot disk

A memory resident driver (MRD) is a program that remains in memory at all times, and modifies the actions of the system. The `buildmrd` program permits the linking together of a number of memory resident driver programs, whilst specifying certain system information that users often like to customise. The result of using this program is a file named `mrdrivers`. The `mrdrivers` file is executed when the system boots, provided it is present in the root directory of the boot disk.

The various MRD programs placed in memory by this means extend the operation of your 1616, by adding new commands to it, or by changing the operation of old commands.

The other major, and perhaps even more important reason for having an `mrdrivers` file, does not involve MRDs. You can configure your system, specifying the ram disk size, the amount of memory for video, the default colours, the number of command lines for the line editor to capture, and so on. This becomes very

important if you have little memory, and need to make the most of it. It is even more important if you have lots of memory, for it vastly extends how you can use that memory. The actual options available when using `buildmrd` are set out below:

`buildmrd [-r`*`ramdisksize`*`] [-o`*`outfile`*`] [-s`*`stacksize`*`] [-v`*`videosize`*`] [-c`*`colours`*`] [-b`*`ncacheblock`*`] [-d`*`dircachegiven`*`] [-f`*`maxfilesgiven`*`] [-l`*`llgiven`*`] [file.mrd] [file.mrd] ...`

Description of options

STACKSIZE

is the amount of space reserved for the system stack in kbytes. If this is not specified it defaults to 32 kbytes. If you are heavily into stacking things, or if you run into programs with insufficient stack space, make it 64k or 96k.

VIDEOSIZE

is the amount of RAM at the top of memory (ending at $80000 on an unexpanded Applix 1616) reserved for video display in kbytes. If not specified this defaults to 32 kbytes (it cannot be made less), usually starting at $78000. If you specify 64 kbytes, for example, it will usually start at $70000. If you will be running EGA video, or mgr, for instance, you need 64 kbytes. You can specify up to a half megabyte of video memory, if you have an expanded Applix 1616.

RDSIZE

is the size of the RAM disk in kbytes, and should be a multiple of 8 kbytes. If not specified this defaults to 200 kbytes. It cannot be made less than 8k. If the `mrdrivers` file cannot be found at boot time, the RAM disk size is set according to the setting of bits 0 and 1 of the hardware diagnostic switches. If you have a memory expansion board, give yourself plenty of ram disk room, up to about a megabyte say.

COLOURS

Colour settings were added to the header structure of the MRDRIVERS file, as at January 1989, with a default setting of 0x1FA50. Bit allocation in this longword is as follows:

00-03	Pallette entry 0
04-07	Pallette entry 1
08-11	Pallette entry 2
12-15	Pallette entry 3
16-19	Border colour

This longword is set using the `-c` flag in `buildmrd.xrel`, when creating the MRDRIVERS file. For example, `-c1fa50` will give a border colour of 1, and pallette entries of 15, 10, 5 and zero (which are actually the default in any case). If the MRDRIVERS file contains zero for the colour setting, the system assumes it is an old version MRDRIVERS file, and sets the colours to the default.

OUTFILE

is the output file pathname. It defaults to `mrdrivers`. Why bother to change it, unless you are generating some additional, altered, `mrdrivers` files? This can be of use if you have multiple, different, mrdrivers files, each intended to do slightly different things with memory.

NCACHEBLOCK

Number of 1 kbyte blocks to devote to cache memory. Don't worry about using this.

DIRCACHEGIVEN

Number of directory entries to cache in memory. Default is 20.

MAXFILESGIVEN

Maximum number of file control blocks open. Just accept the defaults for most things.

LLGIVEN

Number of lines of input to be retained for use by the 'recall' feature of the command line editor. Default is 10, but if you have plenty of memory, increase it wildly! This lets you edit more of the past command lines you used.

MRDFILES

is a list of zero or more memory resident driver files. By convention these filenames end in '`.mrd`'. If no name extension is given, a `.mrd` is automatically added. Programs written for MRD use must comply with the guidelines set out in the *Technical Reference Manual* for memory resident drivers, so you may wish to seek help in writing or converting your first MRD program.

Typical MRDs include time display, screen blanking, and various beep programs. Unfortunately, I haven't organised my disk to use many MRDs, so I can't really comment about using them extensively. Doing a quick check of my system using `find . *.mrd` to search turned up code for the following MRDs.

assign.mrd	Same as `assign` now present in 1616/OS (early version).
beep.mrd	Demonstration that makes a beep sound.
crtc.mrd	Allows you to fiddle with 6545 values on the fly.
crtsav.mrd	Screen saver, turns off video after a preset period, until a key is touched, by Jeremy Fitzhardinge.
except.mrd	Exception handler for bug tracing, by Michael Johnson in SW #5.
locate.mrd	Find data in memory, for program debugging, by Michael Johnson in SW #5
pipe.mrd	Same as | now present in 1616/OS, an early version.

quick.mrd	Shuts down most of video display (and ram refresh) thus speeding up CPU processing by up to 10%.
sseg.mrd	Support for EGA video, from Conal Walsh.
tdos.mrd	Time of day clock in corner of display.

Many of those above were written by Andrew Morton, and are readily available from Applix, either on User Disks, or on Shareware Disks. There are many other MRD programs available. Also, an MRD program is essentially an ordinary program with certain additional facilities added, so many ordinary programs can be turned into an MRD if required. The program `mrdstat` (from Andrew's *Utility Disk*), or Michael Johnson's `readmrd` (on *Shareware #5*) will report on which MRDs you have present in your system.

So, why use an MRD? After all, the 1616/OS is fully multitasking, and you can run tasks in background. In truth, from the user's viewpoint, there isn't a great deal of difference, except for convenience. You generally want a certain combination of facilities; this combination will depend upon what you are doing. So, for programming work, you want MRDs for bug tracing, while for general work with your computer, you may want, say, a time of day clock. The ideal solution is a number of `mrdrivers` files, one for each different type of boot disk.

One day I'll add a list of all known MRD programs in this space, but not this time.

Boot code

I've mentioned that Applix 1616 boot disks include a set of code for starting the system in a particular way. The standard, relatively simple, boot code is in `/f0/sys` and is named `bootv3.exec` or `bootv3`. You will also find the assembler source code in the same directory.

Various users have made changes to the boot code, to allow more freedom to customise the boot procedure.

3
Disk Utilities

In this section of the manual, we describe some of the utilities available on the Applix User Disk. The utilities are described in alphabetical order, rather than the order in which they are used.

arep - keyboard autorepeat rate

```
arep repetition_ticks initial_delay_ticks
```

Description

Allows you to alter the rate at which the keyboard auto-repeats any key that is held down. Each tick is a 50th of a second, and you can set the time between repeats of each letter, and also how long before a held down key starts to autorepeat. The `arep` daemon does occupy some memory, so if you lack memory space, you may prefer not to use it.

Can be a real boon for playing some fast action games. Now, if someone will just do the same thing for the joystick ...

Examples

```
arep repetition_ticks initial_delay_ticks
```

Bugs

Associated files

`arep.c`

See also

kv, scan

Distribution

Early V4 Users Disk /demos

Author

Andrew Morton

bigbuf - set buffer size for character devices

```
bigbuf [bufsize] [satx] [sarx] [sbtx] [sbrx] [cent] [kb]
```

Description

A program that lets you more easily set the size of the character buffer for any of the I/O devices. The default buffer size is set to $4000 (16k bytes), while the minimum allowed is 32 characters.

Of most use when running a serial port in background, or for making up a defacto printer buffer.

Examples

`bigbuf 1024 kb` sets keyboard buffer to 1k.
`bigbuf 4096 cent` sets printer buffer to 4k.

Bugs

Associated files

`bigbuf.c`

See also

Distribution

Users Disk V4.0b /source

Author

Andrew Morton

blockdev - initialising block devices

```
blockdev devname
```

Description

The program `blockdev.xrel` permits the initialisation of block devices, formatting of floppies and the conversion of 1616/OS V2.X disks to 1616/OS V3.*n* and later.

Where `devname` is the name of the device (/f0, /f1, etc) which you wish to format, initialise, etc.

There are three levels of disk preparation available with this program. The most basic level is to simply alter the disk's boot sector program; you enter the name of the new boot program (usually `sys/bootv3`) and this is written onto the disk.

In the next level of disk preparation you may 'initialise' a disk. This recreates the disk's root block, bitmap and directory. All files are lost. The boot block needs to be rewritten after this.

The next level of disk preparation involves a physical format of the disk, followed by initialisation, followed by the boot code setup.

Associated files

```
blockdev.c
bootv3.s
```

Distribution

Users disk

Author

Andrew Morton

bootv3 - boot program for disk

You don’t actually run this - it runs you!

Description

This program occupies a bootable disk’s boot sector, and determines what actually happens when the 1616 is booted. Normally it simply sets the floppy disk step rates, and then tries to run the `autoexec.shell` program.

You can alter this to anything you like, by rewriting the original source code (provided). Make sure it isn’t too large for a single sector on disk. It isn’t intended for beginners to change; it is just something you use.

Examples

Bugs

Associated files

`bootv3.s`

See also

Distribution

User Disk V4.0b /sys

Author

Andrew Morton

bug - early mousecall demonstration

```
bug &
```

Description

`bug &` generates a shape that wanders over the display, in background mode, until cancelled using `kill`.

It can only run asynchronously. You can slow down your display something wonderful if you invoke it a dozen or so times (also makes things fairly unreadable, even though it does restore text after itself). Look at the C code to see how to alter the shape. Notice that the early Version 4 manuals had errors in the description of the mousetrap call. Tells you to buy new roms if you don't have Version 4.

Examples

Bugs

If the screen scrolls while bugs are present, left over bits get stuck to the display. Only if you consider the display as emulating a windscreen, can this be considered a feature!

Associated files

`bug.c`

See also

Distribution

User Disk V4.0b /demo/bug/

Author

Andrew Morton

buildmrd - construct MRDRIVERS file for boot disk

`buildmrd [-r`*`ramdisksize`*`] [-o`*`outfile`*`] [-s`*`stacksize`*`] [-v`*`videosize`*`] [-c`*`colours`*`] [-b`*`ncacheblock`*`] [-d`*`dircachegiven`*`] [-f`*`maxfilesgiven`*`] [-l`*`llgiven`*`] [file.mrd] [file.mrd] ...`

Description

This program permits the linking together of a number of memory resident driver programs, whilst specifying certain system information that users often like to customise. A memory resident driver (MRD) is a program that remains in memory at all times, and modifies the actions of the system. Typical MRDs include the time display, screen blanking, various beep programs.

STACKSIZE

is the amount of space reserved for the system stack in kbytes. If this is not specified it defaults to 32 kbytes. If you are heavily into FORTH, make it 64k or 96k.

VIDEOSIZE

is the amount of RAM at the top of memory (usually ending at $80000) reserved for video display in kbytes. If not specified this defaults to 32 kbytes (it cannot be made less), usually starting at $78000. If you specify 64 kbytes, for example, it will usually start at $70000.

RDSIZE

is the size of the RAM disk in kbytes, and should be a multiple of 8 kbytes. If not specified this defaults to 200 kbytes. It cannot be made less than 8k. If the `mrdrivers` file cannot be found at boot time, the RAM disk size is set according to the setting of bits 0 and 1 of the hardware diagnostic switches.

COLOURS

Colour settings have been added to the header structure of the MRDRIVERS file, as at January 1989, with a default setting of 0x1FA50. Bit allocation in this longword is as follows:

00-03	Pallette entry 0
04-07	Pallette entry 1
08-11	Pallette entry 2
12-15	Pallette entry 3
16-19	Border colour

This longword is set using the `-c` flag in `buildmrd.xrel`, when creating the MRDRIVERS file. For example, `-c1fa50` will give a border colour of 1, and pallette entries of 15, 10, 5 and zero (which

are actually the default in any case). If the MRDRIVERS file contains zero for the colour setting, the system assumes it is an old version MRDRIVERS file, and sets the colours to the default.

OUTFILE
: is the output file pathname. It defaults to `mrdrivers`.

NCACHEBLOCK
: Number of 1 kbyte blocks to devote to cache memory.

DIRCACHEGIVEN
: Number of directory entries to cache in memory. Default is 20.

MAXFILESGIVEN
: Maximum number of file control blocks open.

LLGIVEN
: Number of lines of input to be retained for use by the ‘recall’ feature of the command line editor. Default is 10, but if you have plenty of memory, increase it wildly!

MRDFILES
: is a list of zero or more memory resident driver files. By convention these filenames end in ‘`.mrd`’. If no name extension is given, a `.mrd` is automatically added. These programs must comply with the guideslines set out in the *Technical Reference Manual* for memory resident drivers.

One day I’ll add a list of all known MRD programs in this space, but not this time.

Examples

Bugs

Associated files

`buildmrd.c`

See also

mrdstat

Distribution

Users Disk V4.2

Author

Andrew Morton

chdev - change character devices

```
chdev [device:] [-q] [setting ... ] [setting=VAL ... ]
```

Description

Program allows lots of changes to the default settings of character oriented devices, which include `CON:, SA:, SB: CENT:, NULL:` and `TTY:`. If you are running 1616/OS earlier than Version 4.2, tells you to buy new roms. If running a later version, warns of possible incompatibilities.

This is a (somewhat) easier way of using the extensive range of commands made available in the ***cdmisc*** system call described in the *Technical Reference Manual*.

The **-q** or **-b** option gives a list of possible setting, while the `dev:` device allows changes to standard input (whatever that happens to be).

You can alter or examine the following settings, most of which are for the serial ports SA: or SB:

`baud` *, baud rate (speed) setting of serial ports. Standard values are 300, 600, 1200, 2400, 4800, 9600, 19200, 38400. There may be problems updating with values higher than this, or with non-standard values higher than 2400.

`rxbits` *, number of bits to receive, 5, 6, 7, or 8 (7 or 8 are usual, depending upon the `parity` setting).

`txbits` *, number of bits to transmit, 5, 6, 7, or 8 (7 or 8 are usual).

`parity` *, a check bit, 0 for no parity, 1 for odd parity, 2 for even parity. If `rxbits` or `txbits` are 7 bits, then even parity is usual. If using 8 bits, then no parity is usual.

`stopbits` *, the number of transmitted stop bits sent after each byte. 0 gives 1 stop bit (the usual setting), 1 gives 1.5 stop bits, 2 gives 2 stop bits.

`ll` the number of last lines retained from this device, for the line edit history.

`sigchar`, read or set signal character. The device driver compares incoming characters with the 'sigintchar'. When a match occurs, a SIGINT signal is sent to the process running the device. The 'sigintchar' for the CON: device is normally $83 (Alt Ctrl C). Set to .256 to disable.

`eofchar`, the end of file character. Use 4 (Ctrl D) for UNIX, .26 (Ctrl Z) for CP/M or MS-DOS. Default is .256, or no end of file character.

`resetchar` *, set or read the reset character, and perform a ***warmboot*** system cal if found. The CON: driver uses $92 Alt Ctrl R as the reset character. Use .256 to disable.

`xonchar`, the xon character, used by a device for software handshake. The default for the CON: device is $D1 (Alt Q. The default for a serial device is .17 (Ctrl Q), and this tells the serial device to resume transmitting.

`xoffchar`, the xoff character, sometimes used by a serial device for software handshake. The default for the CON: device is $D3 (Alt S). Set to .256 to disable. The default for a serial device is .19 (Ctrl S), and this tells the serial device to cease transmitting.

`rawmode`, can be set to disable all inout processing. All characters, including SIGINT, resets, xon, xoff are passed along. Can be dangerous, so read the manual first.

`sigint` *, send a SIGINT signal to the process running the device, when it matches an incoming character with the 'sigintchar'. This is how the Alt Ctrl C works.

`sighup` *, send a SIGHUP signal to the process running a device, when it matches an incoming character with the 'sigintchar'.

`killuser`, invoke the ***killuser*** system call upon the process running the device upon receipt of the 'sigintchar'.

`dtr` *, send a DTR (data terminal ready) output signal on pin 9 of the serial port.

`rts` *, send a RTS (request to send) output signal on pin 3 of the serial port.

`dcd`, read DCD (data carrier detect) input line on pin 8 of the serial port.

`cts`, read CTS (clear to send) input line on pin 2 of serial port.

`break` *, set or read the break condition on the transmitter of the addressed device.

`hfc` *, hardware flow control on or off. In the default (on) mode for the serial (SCC) ports, the DCD line (pin 8) qualifies receive date, CTS (pin 2) is used for hardware flow control, DTR (pin 9) is always asserted, while RTS (pin 3) is negated when the device receive buffer is nearly full.

When `hfc` is disabled, both RTS and DTR are asserted by the SCC serial port. The port then runs in three wire (pins 1, 4 and 5) mode, completely ignoring all handshake signals.

`hupmode` * hangup (stop serial port) when DCD line (pin 8) goes low (that is, when the caller hangs up the phone).

`hasmisc`, shows if a miscellaneous entry point vector is installed.

`perms` *, set or read permission bits.

`txcount`, number of characters in serial transmit buffer.

`rxcount`, number of characters in serial receive buffer.

`txroom`, bytes free in transmit buffer before it is overrun.

`rxroom`, bytes free in receive buffer before it is overrun.

`txflush`, flush transmit buffer upon completion of output.

`txpurge`, zero transmit buffer, dumping all pending output.

`rxpurge`, zero receive buffer, dumping all pending input.

`rxpeek`, peek at next character that will be read from the input device by a ***read***, ***getchar*** or similar system call. Returns -1 if not character available.

`version`, tells version number of low-level device driver, or 1616/OS version for device within operating system.

`txbsize` *, current transmit buffer size.

`rxbsize` *, current receive buffer size.

`rxtotal`, channel receive count.

`txtotal`, channel transmit count.

`xlateesc` *, enable Televideo 950 escape code translation to some other set of escape codes.

`rxsigpurge`, purge receive buffer upon receipt of signal.

`txsigpurge`, purge transmit buffer upon receipt of signal.

A number of these commands can only be used by UID 0 (the user at the console). This is to prevent dial-in users from altering sensitive settings. These settings are marked with a *.

Examples

Bugs

Associated files

`chdev.c`

See also

Distribution

Users Disk V4.2

Author

Andrew Morton

chmem - change stack memory usage

```
chmem stackspace files ...
```

Description

Lets you specify the amount of stack space (in k bbytes) to be made available to a list of `.xrel` programs.

As memory is a scarce resource, you may find that the default stack size (usually set by your `mrdrivers` file) is insufficient for certain programs (for instance, the `sort` program will usually occupy additional stack space). If you get an error message suggesting the use of `chmem`, that is a pretty good indication of insufficient stack space.

Conversely, you may wish to decrease the stack size for programs that make little use of the stack. The stackspace is specified in kilobytes.

See the ***runstats 129, 8*** system call in the *Programmers Manual* or *Technical Reference Manual* for checking the actual memory usage of a program.

Examples

Bugs

Associated files

`chmem.c`

See also

Distribution

Users Disk V4.0b /source

Author

Andrew Morton

chmod - change the attribute bits

```
chmod [+rwxabhl] [-rwxabhl] file [files ... ]
```

Description

Lets you more easily modify the attribute bits of files or directories, instead of using the inbuilt `filemode` command. Instead of using a binary mask, you set attribute bits on using the **+** option, or off using the **-** option. The meanings are permit **r**ead, **w**rite, or e**x**ecute. Set **a**rchive, **b**oring, **h**idden or **l**ocked bit.

Read, write, execute and hidden bits do not apply to UID 0 (the user at the console), but only to other UIDs. The archive bit can be used to identify files to be backed up. Locked files are ones that can not be deleted. Boring files are ones that do not ned to be backed up, because they are not altered.

Examples

Bugs

Associated files

`chmod.c`

See also

Distribution

Users Disk V4.2

Author

Andrew Morton

chown - change the ownership of files

```
chown  UID | username  file [files ... ]
```

Description

Lets you more easily modify the ownership of files or directories, by altering the User Identification (UID). Only UID 0 can alter files to UID 0.

`Chown` looks in a file `/etc/passwd` for a user name corresponding to a UID, complains and exits if that user name is not available.

Examples

Bugs

Associated files

`chown.c`
`/etc/passwd`

The structure of the password file is similar to UNIX. It consists of a number of lines, each with a similar structure. Each line contains various fields, separated by a **:** character. For example,
`eric::20:3:eric lindsay:/H0/users/eric:sstools`

The fields are:
Login or user name
Encrypted password (can be blank)
User ID (a long integer number)
Group ID (a long integer number)
Description of user
User's home directory
Default startup program, a shell if nil

See also

chmod

Distribution

Users Disk V4.2

Author

Andrew Morton

diskcopy - make exact copy of disks

```
diskcopy sourcedev destdev [-f] [-r] [-s] [-v]
```

Description

The program `diskcopy.xrel` copies a 1616/OS disk block-for-block to another disk. The DATE and SPECIAL fields in the rootblock of the target disk are altered to prevent the system from assuming the two disks are the same.

Where `sourcedev` is the device identifier of the source disk (/f0, /f1, etc) and `destdev` is the destination disk identifier. If these are the same then you will be prompted to perform disk swaps at the appropriate times. The latest version knows about multiple block copying, and uses it unless the `-s` flag is used. It knows to check for floppy only, and knows difference between memory sizes returned in V3.3 and V4.1. Checks for and will run as an asynchronous (background) process.

`-f` forces physical formatting of the destination disk. This only works if the destination device is /f0 or /f1.

`-r` suppresses the re-randomisation of the root block's DATE and SPECIAL fields, yielding an exact copy of all blocks of the disk.

`-s` forces single block copying

`-v` sets verbose mode: the program prints out more status information as the copy proceeds.

Examples

```
diskcopy /f0 /f1 -v -f
```

This will copy from /f0 to /f1, formatting /f1 and printing out status information as the copy proceeds.

Bugs

Can not handle different sized disks.

Associated files

diskcopy.c

Distribution

Users Disk V4.0b /source

Author

Andrew Morton

fileclean - clean up a file

```
fileclean filename
```

Description

Converts a file into 1616 text format by stripping the top bit, ignoring control characters, and ensuring correct CR/LF sequences.

Examples

Bugs

Associated files

`fileclean.c`

See also

Distribution

User Disk V4.0b /source

Author

Andrew Morton

forever - repeat a command forever

```
forever command
```

Description

`forever command` simply keeps repeating a command, until cancelled using `kill`.

Great for demonstrating that the multitasking really does work.

Examples

Bugs

The first `forever` under `vcon` is hard to kill off.

Associated files

`forever.c`

See also

Distribution

User Disk V4.0b /source

Author

Andrew Morton

fscheck - checking and repairing file systems

```
fscheck devname [-v] [-y] [-yy] [-q]
```

Description

The `fscheck.xrel` utility scans the file system on a disk, reporting any inconsistencies in information on the disk.

`devname` is /f0, /f1, etc.

`-v` causes the program to operate in verbose mode, so more status information is printed.

`-y` automatically answers 'yes' to all its questions, except for the last one where it asks if you wish to write out the disk bitmap; this question must be manually answered.

`-yy` all questions, including the last one, are automatically answered in the affirmative. Automatic file repair, regardless of risk.

`-q` a simplified and much faster check is made. If a file system passes this without complaining, it should be OK. If it complains, then use the other options for more information. Only available from July 1989 on.

`fscheck.xrel` does not check the disk for bad disk blocks - these may or may not be detected, depending upon where on the disk they lie.

Errors detected in operation are repaired by deleting the offending block from the file. The file length is reduced by 1024 bytes (unless the block is at the end of the file). A message is displayed and you are asked whether or not the fix is to be made. The file will probably be corrupted. If a file's block map block number is an invalid block then the file is deleted. If a directory's block range is invalid then it is removed. The user is told about all of these things and prompted to confirm. A number of these fixes will cause bitmap inconsistencies, but these are fixed later or on another pass of the file system check program.

This program should be rerun until it gives the disk a clean bill of health; it may take several passes to fully repair a disk.

If a change is made to the bitmap of a non-removable disk this program exits with a ***warmboot*** system call, which is like pressing reset. This must be done because the system never re-reads the bitmap and root block of a non-removable device, so alterations to the bitmap would not be noted if this program exited in the normal manner.

Examples

```
fscheck -v -yy /f0
```

Bugs

Takes forever on a large file system, such as a hard disk (take the `yy` option and go to bed). Probably will die on a really large file system (say 40 megabytes).

Associated files

`fscheck.c`

Distribution

V4.0b User disk /bin

Author

Andrew Morton

genreloc - produce relocatable format file

```
genreloc fname1 fname2 ofname [bsssize]
```

Description

This program reads and crunches two otherwise identical `.exec` files (fname1 and fname2), the execution addresses of which differ by the magic number 0x20002, to produce a relocatable format output file (ofname), with an `.xrel` extension. You can optionally specify the bsssize.

It is used because the built in `ssasm` assembler can not directly produce `.xrel` files (yes, I know about the -r option, and will tell you when it actually works properly). When a program you are developing is complete and tested, use the `makexrel.shell` program. This simply runs your code through the assembler twice, with an `ORG` that differs by 0x20002, and invokes `genreloc` to produce the final `.xrel` file.

Note that `genreloc` may complain about stack space. Use `chmem` to fix it. `genreloc` will locate phase errors (that is, the two programs weren't identical), and warn if it finds no relocations (possible lack of ORG statement error). It reports the number of bytes of code, and the number of relocations.

Examples

`makexrel filename` (use the shell, to avoid typing long commands).

Bugs

Always has the wrong stack size. Use `chmem` to fix that.

Associated files

`genreloc.c`

See also

`makexrel.shell`

Distribution

User Disk V4.0b /source or
Utilities/lisasource

Author

Andrew Morton

ibmfont - replace screen font with IBM lookalike

`ibmfont`

Description

Reload the 1616 screen font with an IBM PC compatible font. The source contains the expected vast number of character bit maps, so it is simple, but long.

Examples

Bugs

Associated files

`ibmfont.s`

See also

Distribution

User Disk V3 /sys /bin

Author

Andrew Morton

if - command line options

```
if command then command [else command]
```

Description

Why give up just because a command returns an error. Use this instead! Check out the `test` program also.

Examples

Bugs

Associated files

`if.c`

See also

Distribution

User Disk V4.2

Author

Andrew Morton

killpg - signal a process group

```
killpg [-nn] process_group [process_group ... ]
```

Description

Send a signal to a process group. *-nn* is the signal number to send. If no signal is specified, the ***kill*** is sent as a default.

The signals available are described in the include file `signal.h`, and are similar to UNIX. Not all are appropriate.

1	sighup	hangup line
2	signit	interrupt ^C or break
3	sigquit	quit ^\
4	sigill	illegal instruction
5	sigtrap	trace trap
6	sigiot	IOT instruction
7	sigemt	EMT instruction
8	sigfpe	floating point exception
9	sigkill	kill, cannot be caught or ignored
10	sigbus	bus error
11	sigsegv	segmentation violation
12	sigsys	bad argument to system call
13	sigpipe	write on pipe with no-one to read
14	sigalrm	alarm clock
15	sigterm	software termination signal from kill
16	sigusr1	user defined signal 1
17	sigusr2	user defined signal 2
18	sigcld	death of a child
19	sigpwr	power fail restart (not used)
20	sigstop	suspend process
21	sigcont	restart process
22	sigexit	someone exited
23	sigblockrx	someone sent us an IPC block

Examples

Bugs

Associated files

`killpg.c`

See also

Distribution

User Disk V4.2

Author

Andrew Morton

loadreloc - convert .xrel file to .exec file

```
loadreloc ifname.xrel
```

Description

Reads `ifname`, an `.xrel` file, into memory at $8000, and then converts it into an `.exec` style program. I'm not at all sure why Andrew needed this one. Probably left over from the old days.

Examples

Bugs

Associated files

`loadreloc.c`

See also

Distribution

Utilities/lisasource

Author

Andrew Morton

mem - memory useage daemon

```
mem logfile
```

Description

`mem logfile` is a memory usage daemon, used for checking how much memory is being used.

Examples

Bugs

Associated files

`mem.c`

See also

Jeremy Fitzhardinge has done a much more involved version.

Distribution

Early V4 User /demo

Author

Andrew Morton

mrdstat - details of MRDs resident

`mrdstat`

Description

`mrdstat` dumps out information about the current memory-resident driver installation. Latest version gives details of the magic number, ram disk size, memory usage, stack space allocated, video ram allocated, mrd start address, ram disk start address, bitmaps, default colour palette entries, number of last lines allocated, number of file cache blocks, and directory cache entries.

Very handy when you can't recall what the hell is currently loaded in memory from your most recent boot.

Examples

Bugs

Associated files

`mrdstat.c`

See also

Distribution

User Disk V4.2

Author

Andrew Morton

newsetal - test alias system call

```
alias [on] [off] [exit] [list] [max_alias]
```

Description

Place logfile in `/usr/lib/alias`
Won't work without 1616/OS V4.2 or later. Allows up to 100 entries. Must run asynchronously. Includes lockin (reentrant protection) in code. I couldn't for the life of me work out what Andrew is doing with this one! Couldn't work out the (undocumented) syscall either.

Examples

Bugs

Associated files

`newsetal.c`
`/usr/lib/aliases` (the log file)

See also

setalias

Distribution

User Disk V4.2

Author

Andrew Morton

nice - set CPOU use of live process

```
nice nice_level [pid] [pid ... ]
```

Description

Set the nice level of a running process. Reduce the number of time slices a process gets, so that it takes up less of the CPUs effort. For instance, setting a lower priority for printer spooling could be a good move, since printers are generally too slow to keep up anyway.

The available levels are 0 to 9.

Examples

Bugs

Associated files

`nice.c`

See also

Distribution

User Disk V4.2

Author

Andrew Morton

pcrip - shoot the IBM game

`pcrip`

Description

`pcrip` is a game where you get to shoot moving IBM computers. Note that XT's move slowly, AT's move faster, and PS/2's move with a jerk (take that however you like) because of the multitasking. Good for testing your joystick (mine is utterly destroyed).

If it is too frustrating, try Jeremy Fitzhardinge's version (probably under the same name), with far more PCs to shoot, and a shotgun, instead of single shots.

Examples

Bugs

Associated files

`pcrip.c`

See also

Distribution

User Disk V4.0b /demos/pcrip

Author

Andrew Morton

ps2 - process status

`ps2`

Description

`ps2` indicates the status of all processes currently running under 1616/OS. It will complain if you attempt to run it on a non-multitasking version of 1616/OS. Looks very similar to the inbuilt `ps` command, but at least you get to read the code and find out how it was done.

There are numerous variations, from Andrew and others, floating round giving slightly different information, and the code is easy to change.

Examples

Bugs

Associated files

`ps2.c`

See also

`ps`

Distribution

User Disk V4.0b /source

Author

Andrew Morton

rawcopy - copy Minix or other disk

```
rawcopy sourcedevice destdevice
```

Description

Use low level commands to copy a foreign file system disk, such as a Minix file system.

Examples

Bugs

Associated files

No source provided.

See also

Distribution

User Disk V3 /bin

Author

Andrew Morton

readuid - read out the user ID number

`readuid`

Description

A short program to read the User ID number of another user.

Examples

Bugs

Associated files

`readuid.c`

See also

setuid

Distribution

User Disk V4.2

Author

Andrew Morton

relcc - C compiler preprocessor - SW#6

```
relcc [-acdijlorsuv] file
```

Description

`relcc` is a preprocessor to the Hitech C compiler. It allows you to produce `.xrel` relocatable programs. You use it instead of the HiTech C compiler, however you MUST have bought a copy of all the rest of the HiTech C before this will do anything.

-a

-c Don't link, use the `.obj` file.

-d

-h Write out `l.sym` symbol table information in the current directory, for use by the Minix debugger (I think).

-i Location of the include files, typically `-i/hitech/include`.

-j Specify `-jmrd_crtapp.obj` in command line, if attempting to produce an MRD driver.

-l Libraries to be added

-n Lets you capture output without compiling.

-o Optimise, if invoked.

-r Disable relocatable (`.xrel`) output if invoked. Code produced is in `.exec` form.

-s

-u

-v Verbose mode if invoked.

Disks should be set up with the following `assigns`.
`assign /temp /rd` OR
`assign /temp .`
`assign /hitech /h0/hitech OR`
`assign /hitech /f0/hitech` as appropriate.
Note that `libc.lib, libf.lib` etc., should be placed in the `/hitech` directory.

Don't ask me; I couldn't follow what half the code was doing.

Examples

Bugs

Associated files

`relcc.c`, `kmem.c`

See also

c.c (on Hitech C compiler disk)

Distribution

User Disk V4.0b /source, SW#6

Author

Andrew Morton

renlower - rename files in lower case

```
renlower [file] [file] ...
renupper [file] [file] ...
```

Description

`renlower filename` reads the names of files from the command line, or from standard input, and changes them from upper case to lower case. It accepts wildcards, and also changes the names of directories.

Can be used with pipes from the output of `find` to locate all examples of a particular style of filename, in any directory.

`renupper` is a basically identical program, slightly reworked to have the opposite effect.

Examples

Bugs

Associated files

```
renlower.c
renupper.c
```

See also

Distribution

User Disk V4.0b /source

Author

Andrew Morton

setalias - set an alias

```
setalias [max_alias]
```

Description

Possibly an earlier version of `newsetal`, listed earlier. Allows you to set a maximum of 10 aliases.

Examples

Bugs

Associated files

`setalias.c`

See also

Distribution

User Disk V4.2

Author

Andrew Morton

setenv - set an environment for a user

`setenv [-q] [-v]` List long help
`setenv ENVVAR=SETTING` Adds a setting
`setenv ENVVAR` Deletes setting

Description

The `setenv` program allows us to manipulate the user's environment, in a manner similar to that provided by the inbuilt `option` command. When running multiple users, it has the advantage that many different environments (one per user) can be established.

The ENVVAR item is replaced by any of the following terms, whose meanings are given below.

prompten	Enable working directory in prompt
verbose	Enable verbose mode
dirmode0	Sort directory by date
dirmode1	Sort directory by name
ibbeep	Enable bell on command error
nobak	Disable `.bak` file generation by editors
errbeep	Enable bell on all errors
hidefiles	Hide files with hidden bit set
assignprompt	Use assigns to shorten prompt
lowernames	Enable lower case file names
promptgt	Enable > in prompt
dodsign	Do **$** exec environ substitutions
doarg	Do all argument environ substitutions
doarg0	Do command name environ substitutions
noshellout	Deny shell escapes from applications

Complains if you don't have at least Version 4.2 EPROMS. Expects to have a file `/usr/lib/environ` on the current device as the location of the default environment details.

Examples

Bugs

Associated files

`setenv.c`
`/usr/lib/environ`

See also

Distribution

User Disk V4.2

Author

Andrew Morton

setuid - start a shell for new user

`setuid -u`*`UID`* `-m`*`UMASK`*

Description

Start a shell for a new user, specifying the user number (1 to 64k), and the user file permissions mask.

Examples

Bugs

Associated files

`setuid.c`

See also

umask

Distribution

User Disk V4.2

Author

Andrew Morton

sset - front end for set command

```
sset [-l] name[=setting] [name=setting ...]
```

Description

A front end for the rather complicated `set` command, used to arrange replacement of string variables by other string settings, and save the new settings in a file `$(HOME)/settings.shell`.

You should arrange to modify your `bootv3` boot file so that `autoexec1.shell` and `autoexec2.shell` files are executed upon Level 1 and Level 2 resets. Both these autoexec files should then execute the `$(HOME)/settings.shell` file, so as to restore your previous environment strings.

`sset -l` reloads all settings from the `$(HOME)/settings.shell` file.

You should set HOME=/pathname/home_directory. Some environment strings have special meanings. For example, ‘home’, ‘path’, ‘lowbaud’. If I ever get round to figuring it out, I’ll document it.

Examples

Bugs

Associated files

```
sset.c
$(HOME)/settings.shell
```

See also

Distribution

User Disk V4.2

Author

Andrew Morton

start - start a process
stop - stop a process

```
start pid [pid ...]
stop pid [pid ... ]
```

Description

Stop and start processes, without throwing them out of memory, and without the need to reload from disk.

Examples

Bugs

Associated files

```
start.c
stop.c
```

See also

Distribution

User Disk V4.2

Author

Andrew Morton

test - give information about files

```
test command filename
```

Description

A general purpose test for the nature of files. There are six forms:

```
test + - isthere  filename
test + -  isdir  filename
test + -  isolder  filename1 filename2
test + -  islarger  filename1 filename2
test + -  hasextent filename.extent
test + -  error command
```

Examples

Bugs

Associated files

```
test.c
```

See also

if

Distribution

User Disk V4.2

Author

Andrew Morton

tstint - test whether interactive

`tstint`

Description

Uses ***syscall 129,16*** to determine whether a program is interactive or not, and whether it can run in background.

Examples

Bugs

Associated files

`tstint.c`

See also

Distribution

User Disk V4 /demos

Author

Andrew Morton

trace - demonstrate trace mode

`trace`

Description

Demonstrates assembler trace mode stepping.

Examples

Bugs

Associated files

`trace.s`

See also

Distribution

User Disk V4 /ssasm_srce

Author

Andrew Morton

tune3 - play music

tune3 `musicfile`

Description

Plays musical notes, according to a list in a music file. The macro capabilities of the `ssasm` assembler are used to turn a text file into a format suitable for playing using `tune3`. The text files are normally given the extension `.mus`, but must have a `.s` extension prior to being assembled. The resulting `.exec` file is renamed to get rid of the `.exec` extension prior to being played.

`copy roncalli.mus yourfile.s`
(This makes a copy in assembler format)
`edit yourfile.s`
(Modify to the tune you require)
`ssasm yourfile`
(Assemble the file, including macros)
`rename yourfile.exec yourfile`
(Get rid of the `exec` suffix)
`tune3 yourfile`
(Play the tune)

Now you will need to know just what you should do when editing. The keywords used in editing music files are as follows:

TEMPO	Specify a number for the slowness of the piece, e.g.3000.
DECAY	Part#, value. part# is 1,2 or 3. Value is 1 (long) to 4 (short).
VOLUME	Part#, value. Values can range from 0 (off) to 255 (full).
CHORD	Duration, Pitch1, Pitch2, Pitch3 Duration is 1 (quave), 2 (crotchet), 4 (minim), 8 (semibreve) etc. Pitch is 1 (C below bass clef), and increasing in semitones to about 50.

Modifications (notes by Lindsay Scales)

To make the notes easier to write, and more readable, make these simple modifications to provide note names (these are a subset of the MIDI range). Edit `compose.mac` to add the contents of `musicdefs.mac` and `compose2.mac`. Then add the following equates, where:

N specifies a musical note (to prevent confusion with address and data register equates).

C is the note name (A, B, C, D, E, F, G).

F or **S** is added to indicate sharps or flats.

3, 4, 5, 6 is the octave number, spanning from A flat to G sharp.

eg.

```
NC3     EQU     1       Lowest note in Tune3
NCS3    EQU     2       Provide both a sharp
NDF3    EQU     2       and flat name for note
ND3     EQU     3
...     ...     ...     more notes
NGS3    EQU     9       Octave changes after G sharp
NAF4    EQU     9       Another octave, and repeat ...
...     ...     ...
```

Finally, change the *Include* statements in the `.s` (`.mus`) file to: `include compose.mac`. Other equates could also be used, such as `long equ 1` for note decays, `crotchet equ 2` for note values, and `ff equ 255` for note loudness.

This system of macro expansions would be quite valuable for other applications, where data files need to be knocked up quickly, such as MIDI data, bit mapped graphics file headers, setup files for printers, etc.

Quibbles

Andrew didn't document this.

Associated files

`ssasm.xrel, tune3.xrel, musicdefs.mac, compose2.mac`
Music files by Andrew: `roncalli, bach`
Music files by Craig Mills: `duel, green, maid, saints, short, stairway, wedding` on SW#9 /mills

See also

sound, play, etc.

Distribution

`Tune3` on various User Disks.
Music files by Craig Mills on SW#9 /mills

Author

Andrew Morton.

uid - display and change user identification

```
uid
uid n
uid n files
uid n -
```

Description

`uid` displays the current user ID (default is 0).

`uid` *n* changes the current UID to *n*.

`uid` *n* `files` changes the UID on the specified list of files.

`uid` *n* `-` reads a list of filenames from standard input, and changes the UID on each of them.

User IDs are a number, from 0 (unrestricted) to 65535. You can restrict file access on dial up lines by using them.

Examples

Bugs

Associated files

`uid.c`

See also

Distribution

User Disk V4.0b /source /bin

Author

Andrew Morton

umask - set user file permissions mask

```
umask -albhrwx [-] [-v] [file] [file] ...
```

Description

`umask` eases the task of setting file permissions. Accepts a list of filenames on the command line, or reads from standard input.

The options include

a Archived or backed up

l Locked against deleting

b Boring, don't bother to archive

h Hidden, if required by environment

r Read permissions given

w Write permissions given

x eXecute permissions given

- read file names from standard input

v Verbose mode

Examples

Bugs

Associated files

`umask.c`

See also

Distribution

User disk V4.0b /source

Author

Andrew Morton

vcon - virtual console, video context switcher

`vcon`
`vconp`

Description

`vcon`

Use the Alt 0 keys, the Alt 1 keys, the Alt 2 keys, etc., to change from screen to screen.

`quit` in **all** windows to terminate the program.

Two versions, one does multiple full screen switches, the other puts three smaller windows on a single display.

Tells you to buy new roms if you try to run it without having 1616/OS Version 4. Must run synchronously, **not** in background.

Examples

Bugs

Associated files

`vcon.c`

See also

Distribution

User Disk V4.0b /demos/vcon

Author

Andrew Morton

vcon2 - new virtual console, video context switcher

`vcon2`

Description

`vcon2`

Use the Alt 0 keys, the Alt 1 keys, the Alt 2 keys, etc., to change from screen to screen. Use Alt S to stop a process.

`quit` in **all** windows to terminate the program.
Can not be run in asynchronously, in background.

New versions, does multiple full page window changes.

Tells you to buy new roms if you try to run it without having 1616/OS Version 4. Must run synchronously, **not** in background.

Examples

Bugs

Associated files

`vcon.c`

See also

Distribution

User Disk V4.0b /demos/vcon

Author

Andrew Morton

arep repetition_ticks initial_delay_ticks
bigbuf [bufsize] [satx] [sarx] [sbtx] [sbrx] [cent] [kb]
blockdev devname
bug &
buildmrd [-r*ramdisksize*] [-o*outfile*] [-s*stacksize*] [-v*videosize*] [-c*colours*] [-b*ncacheblock*] [-d*dircachegiven*] [-f*maxfilesgiven*] [-l*llgiven*] [file.xrel] [file.xrel] ...
chdev [device:] [-q] [setting ...] [setting=val ...]
chmem stackspace files ...
chmod [+rwxabhl] [-rwxabhl] file [files ...]
chown UID | username files [files ...]
diskcopy sourcedev destdev [-f] [-r] [-s] [-v]
fileclean filename
forever command
fscheck devname [-v] [-y] [-yy] [-q]
genreloc fname1 fname2 ofname [bsssize]
ibmfont
if command then command [else command]
killpg [-nn] process_group [process_group ...]
loadreloc ifname.xrel
mem logfile
mrdstat
alias [on] [off] [exit] [list] [max_alias]
nice nice_level [pid] [pid ...]
pcrip
ps2
rawcopy sourcedevice destdevice
readuid
relcc [-acdijlorsuv] file
renlower [file] [file] ...
renupper [file] [file] ...
setalias [max_alias]
setenv [-q] [-v] ENVVIR [SETTING] ...
setuid -u*UID* -m*UMASK*
sset [-l] name[=setting] [name=setting ...]
start pid [pid ...]
stop pid [pid ...]

```
test + - isthere  filename
test + -  isdir  filename
test + -  isolder  filename1 filename2
test + -  islarger  filename1 filename2
test + -  hasextent filename.extent
test + - error command
tstint
trace
tune3 musicfile
uid
uid n
uid n files
uid n -
umask -alrwx [-] [-v] [file] [file] ...
vcon
vconp
vcon2
```

5
Error Messages Appendix B

Most of the error messages which come out of 1616/OS are self explanatory. There is an amount of internal consistency checking and protection in version 3 and 4 of the OS, and violations of these can produce error messages which need more interpretation.

Block and memory errors

Error code		Meaning
-1	ffff ffff	Unknown, general error
-2	ffff fffe	Disk is write protected
-3	ffff fffd	Invalid block driver number on block I/O system call
-4	ffff fffc	No space to install block device driver (8 max)
-5	ffff fffb	I/O error: bad disk, door open, RAM disk block checksum failure
-6	ffff fffa	Invalid block requested on block I/O call
-7	ffff fff9	Disk full
-8	ffff fff8	Invalid file handle on file I/O system call
-9	ffff fff7	Bad file name
-10	ffff fff6	File full (formerly 512k, obsolete)
-11	ffff fff5	File not currently open: possibly a bad file handle
-12	ffff fff4	File not open for reading
-13	ffff fff3	File not open for writing
-14	ffff fff2	File open for reading (not writing)
-15	ffff fff1	File open for writing (not reading)
-16	ffff fff0	Out of FCB’s - too many files open (16 max)
-17	ffff ffef	File not found
-18	ffff ffee	Attempted to read beyond end of file
-19	ffff ffed	Duplicate filename would result from ***rename***
-20	ffff ffec	Invalid argument: some nonsense passed to a file I/O call
-21	ffff ffeb	Attempted to seek beyond end of file
-22	ffff ffea	Seek not allowed - file is write-only (obsolete)

-23	ffff ffe9	File currently open
-24	ffff ffe8	Memory allocation failure: out of memory
-25	ffff ffe7	Directory: file-only operation performed upon a directory
-26	ffff ffe6	Not directory: directory-only operation on a file
-27	ffff ffe5	Tried to delete a non-empty directory
-28	ffff ffe4	Directory is full
-29	ffff ffe3	File is locked
-30	ffff ffe2	Bad magic number in relocatable file header
-31	ffff ffe1	User interrupt killed program
-32	ffff ffe0	No permission for file access
-33	ffff ffdf	Exec level too deep
-34	ffff ffde	Too many processes
-35	ffff ffdc	Exited due to kill
-36	ffff ffdb	Read/Write pipe with other end closed
-37	ffff ffda	Invalid PID passed to system call

These other error messages fall into two categories: internal errors, and warnings.

Internal errors

The system responds to an internal error by printing the following message:

Internal error N1: call PC = $N2. N3 N4

Where N1 is the internal error number, N2 is (perhaps) the program counter value at which the internal error occured. N3 and N4 are additional information, presented as hexadecimal numbers. After this message is displayed, the system halts, and needs to be restarted.

If an internal error can be reproducibly and inexplicably generated, please send APPLIX details of how to produce the error, so it can be investigated.

The implemented internal errors are listed below:

Internal error 1 and 2

Inconsistency in the memory manager (***getmem***), possibly due to memory corruption.

Internal error 3, 4, 5 and 6

Inconsistency in the memory manager (***freemem***), possibly due to memory corruption.

Internal error 7

Inconsistency in the memory manager during automatic freeing of a program's memory on return from ***exec***.

Internal error 10

A request was made to ***getmem*** for more memory than was available. This internal error may be disabled with OPTION 5. If OPTION 5 is disabled, ***getmem*** returns an error code to the calling program when out of memory, rather than generating an internal error.

Internal error 100, 101 and 102

Problems involving memory allocation and freeing in the line editor last-line recall management. Possibly caused by memory corruption.

Internal error 400

The file system code decided to deallocate a block which lies beyond the range of the disk. Probably caused by a corrupted file system. Use `fscheck.xrel` to repair it.

Internal error 500

In the ***callmrd*** system call a bad MRD header was encountered. This means that the memory reserved for MRDs has been corrupted. The system must be cold booted to recover.

Warning messages

These messages are produced when something unpleasant has been detected.

Suppressed write to block N1 on DEV [N2]

This message comes out of the ***blkwrite*** system call when an attempt has been made to write to the system block N1 on the device identified by DEV. The second number $N2 is the address from which the block was intended to be written. The system disables writing to the system blocks unless OPTION 8 has been set.

Panic: out of memory. PC = $N1

Yes, well. A call to ***getmem***, with OPTION 5 turned off, returned an error message during an attempt to allocate storage for the current directory cache. Solder in more RAM chips.

Released block N1: already free

During an attempt to deallocate block number N1 on a file system, it was discovered that the block was already free. This indicates an inconsistency in the file system. Run `fscheck.xrel` on the disk immediately to attempt to repair the disk.

Odd load address

The system was asked to load an `.exec` file to an odd address (68000 processors don't execute code at odd addresses).

Bad header magic

The magic bytes in the header of an `.xrel` file were not present. Caused by a misnamed or corrupted file.

Truncated xrel file

Something is wrong with the relocation table at the end of an `.xrel` file.

freemem(N1)

Somebody called ***freemem*** with an argument of N1 (N1 is in hexadecimal). The address N1 does not correspond with the memory manager's information, so either N1 was never returned by ***getmem*** or ***getfmem***, or some corruption of memory has occured.

freemem(0xN1)[OxN2] returns 0xN3

The system attempted to free some memory which it used for internal purposes, and ***freemem*** returned the error value N3. The system called ***freemem*** from address N2. The address which was being freed was N1. Caused by corrupted memory.

Corrupted MRdrivers

Produced at the same time as internal error 500, described above.

Booting from /..

Produced when a reset fails to find a suitable boot block on any drive.

Header checksum error

Error in a block device header checksum.

Found blocksize too large

Produced when tape read fails.

Switch 2 open, using SA: for console

Console is serial port, only produced if disk co-processor card is not present, and DIP switch is set for serial port as console.

CON SA SB CENT NUL

List of available stream device names.

System closing pathname

A transient program left a file open when it terminated. The system closes this file and prints a warning message. The autoclosing of files was disabled by `option 7` in Version 3.

Index

Table of Contents