Applix
Utilities

Disk # 1

Applix 1616 Software

UNIX style disk and file utilities
converted or written
by Andrew Morton

Applix 1616 Utilities
Disk One

Disclaimer

None of us claim that these utility programs are good for anything. If **you** think they are, great, but that is up to you to decide. If any, or all, of these programs don't work, that is your problem, not ours. If you lose a million dollars, or anything else, because one or all of these programs stuffs up, you are out of pocket the million, not us. If you don't like this disclaimer: tough. We reserve the right to do the absolute minimum provided by law, up to and including nothing.

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.

This disclaimer has been provided in plain English, in keeping with Applix's policy of providing comprehensive, readable information about its products. It is basically the same disclaimer all the fancy, expensive overseas software packets provide, but without legal beagles mangling the English. Special thanks to Dave Horsfall for bringing this disclaimer to my attention.

Parts of the preliminary version of this manual were based on umpteen different UNIX manuals and textbooks.
It was rewritten to correspond somewhat with the actual source code 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 BBS systems (ringback) on (02) 554 3114 and (02) 540 3595

ISBN 0 947341 ?? ?

Introduction

This manual attempts to describe a collection of disk based utility programs written or converted by Andrew Morton, designer of the Applix 1616 computer system.

A number of these programs are more or less loosely based upon UNIX utilities. This means that, like many other programs within 1616/OS, they are intended to be used in groups. They are tools for building more elaborate programs, with the output from one program becoming the input to another.

The programs can not all be considered complete, polished, commercial products. Many are simple conversions of Public Domain products, where conversion attempts ended as soon as they compiled and ran the first time. There may well be horrendous bugs still in them. Others were written simply to solve some simple problem, and never intended for widespread use. They are tools, and sometimes rather rough round the edges. If you can't live with that, don't buy them! If you do buy them, don't complain!

The good side of all this is that C source code is provided (unlike commercial programs). If you want to alter the way something works, you can do so (provided you have bought the HiTech C Compiler).

What Applix do undertake to provide for your paltry $29.95 fee is a diskette in Applix **1616/OS Version 3** (or later) format that was readable on at least one Applix 1616 system. In the event of a dispute about whether it contains what we say it contains, Applix also undertake to recopy the programs, and demonstrate that the diskette can actually be read on a properly working Applix 1616 system. If it still doesn't work on your system, it probably means you built your system wrong, or didn't upgrade it to Version 3, and that is your problem (see the disclaimer at the start of the manual).

The diskette contains approximately 30 compiled or assembled programs in `.xrel` form, together with source code for the majority of these programs. The source code is generally similar to that actually used to generate the `.xrel` programs.

Applix also provide a printed manual of at least as many pages as there are programs. The manual was written by reading the source code, and describing what I thought the program should do. If I couldn't make any sense of the code, I described what the corresponding UNIX program should do. In some cases, parts of the manual were rewritten to correspond with what the program actually ended up doing when I ran it. Special thanks to Philip Hutchison, Matthew Gardner and Dave Wilson for corrections.

I would be delighted to hear from users with corrections or additional details of how any of these programs work. Any changes will be incorporated in later versions of this manual.

addcr - add carriage returns to a file

addcr `infile outfile`

Description

Reads through the input file looking for newline characters, and adding a carriage return whenever it finds one. Places the result in the output file.

Used to help convert text files from other computers or editors. See the `-a` option in Greyham Stoney's disk software for automatic conversions from MS-DOS.

For fixing files that have unknown carriage return and line feed problems, run them through `rmcr` first (to leave only line feeds), then through `addcr` (to make up CR-LF pairs).

Bugs

Does not accept redirection.

Associated files

`addcr.c`

See also

rmcr, **tolower**, **toupper**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

ar - archive maintainer

ar [**-adprtvx**] `afile` [`filename`] ...

Description

The archive maintains groups of files combined into a single archive `afile`. Individual files are inserted without alteration into the archive file. Maintains the dates files were added. Handy for keeping track of a group of files with some common purpose, such as C programs comprising a larger program. Not the same as `arc`.

The options are a **-**, followed by one character from the set **adprtvx**. `Afile` is the archive file. The `filenames` are constituent files in the archive file. The meanings of the option characters for operations on an archive are:

a Append a file to the end of the archive file.

d Delete the named files from the archive file.

p Print the named files from the archive.

r Replace the named files, or add a new file to the archive, if the replaced file does not exist. **Ar** will create `afile` if it does not already exist.

t Print a table of contents of the archive file. If no names are given, all files in the archive are described. If names are given, information about only those files appears.

v Verbose. Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with the **p** option, the verbose option causes **ar** to print the key letter and file name associated with each file for that operation. For the **r** operation, **ar** will show an "**a**" if it added a new file, or an "**r**" if it replaced an existing one.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter (i.e. delete entries from) the archive file.

Examples

```
ar -r newlib.a f3 f2 f1 f4
```

will create a new file (if one does not already exist) in archive format with its constituents entered in the order shown in the above command line.

Associated files

`ar.c`

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Michael Huisjes

Conversion to Applix 1616 by Andrew Morton

arc - compress and archive files

arc [**-**]{amufdxeplvtc} [biswn] [g*password*] *archive* [*filename ...*]

Description

arc archives, compresses and extracts files. Often uses to collect related files into a single file prior to transmission electronically.

a add files to archive.

m move files to archive.

u update files in archive.

f freshen date of files in archive.

d delete files from archive.

x extract files from archive.

e extract files from archive.

p copy archived files to standard output.

l list files in archive.

v verbose listing of files in archive.

t test archive integrity.

c convert entry to new packing method.

b retain backup copy of archive.

i maintain IBM PC compatible archive.

s suppress compression (store only).

w suppress warning messages.

n suppress notes and comments.

g encrypt decrypt archive entry.

Associated files

? Don’t know where the source is.

Distribution

Applix Utility Disk #2

Author

Ported to 1616 by Andrew Morton.

at - run a task at a specified time

```
at HHMM [command] or
at MMDDHHMM [command]
```

Introduction

at is a program which permits you to run a group of one or more 1616/OS commands at a specified time up to a year hence. at assumes the use of cron, which actually does the work of starting the at job at the correct time.

What at does is to create a file containing a 1616/OS directory name and a set of 1616/OS commands in the directory /usr/lib/atjobs. This file's datestamp is the date and time at which the at job is to be run. cron regularly runs a program called atrun which looks to see if any of the files in /usr/lib/atjobs have a datestamp older than the current time. If they have atrun executes the commands in the file and deletes it.

Because atrun runs jobs at any time after they are due it may be used for reminders. Running an at job on a particular date will result in the job being executed at the first possible time after that date, shortly after cron is started.

To run at you must have a line of the form

```
; Run 'at' jobs every 15 minutes
0 0,15,30,45 * * * atrun /usr/lib/atjobs/at.* or
0 0,15,30,45 * * * atrun -l /usr/lib/atjobs/at.* or
0 0,15,30,45 * * * atrun -l/f0/atlog /usr/lib/atjobs/at.*
```

in your crontab file. at records the current working directory in the at job file, so atrun can change there when it runs the job. This means that relative pathnames will work correctly.

Usage of at is used as follows:

```
at HHMM [command] or
at MMDDHHMM [command]
```

where HHMM represents the hours and minutes of the time when the command is to be run. If this is earlier in the day than the current time at assumes that it is to be run tomorrow. MMDDHHMM represents the month, day, hour and minute of execution. If this represents a time before midnight the previous day then it is assumed to be next year.

If the 'command' is not given at will enter an interactive mode where one or more commands may be typed into the at job. They will be executed synchronously, in order, at the specified time. If the command to execute is supplied on the command line be careful to quote any redirections or wildcards. Note that atrun runs under cron, with its settings of standard input, output and error, so any output from at jobs will come out on cron's output, unless redirected.

The at jobs are recorded in plain text files in /usr/lib/atjobs/at.* and can be read, deleted or altered.

Usage of atrun

atrun is used as follows:

```
atrun [-l[logdirectory]] filenames
```

This program inspects the timestamp on all the passed files and for all those files which are older than the current time atrun opens them, reads the first line, does a ***chdir()*** system call to the directory whose name is contained in that line, and then reads lines one at a time from the file, executing them via the ***exec***() system call.

If the -l flag is given atrun places the output from each ‘at job’ file in a uniquely named file in the directory /usr/lib/atlog. The output directory may be altered by putting the name of the desired directory hard up against the -l, with no white space separator. If atrun is run under 1616/OS V4.1 only the standard output from atrun will be placed in the log file. This is due to a slight OS bug, which should be fixed under 1616/OS V4.2. The log files contain some extra information about what command was run and when it ran, etc. Simply typing these out will tell you what happened and when.

If -l is not given, the commands which atrun reads from the ‘at job’ file will, when executed, inherit atrun’s standard input, standard output and standard error. If these are not redirected in the atrun command entry in the crontab file which started atrun up then they are inherited from cron itself.

Example at commands

1) To delete all editor files at the next occurrence of 1 AM:

```
at 0100 > find /h0 "*.bak" | mexec "delete %s" >null: }null:
```

2) To remind yourself to go to work:

```
at 0800 "echo ^GGo to work >con:"
```

3) To remember a birthday:

```
at 07150000 > echo "It is AKPM’s birthday: send cheque or money order"
>con:
```

Note that this at job will be executed the first time you run cron after 00:00 on the next occurrence of July 15.

Andrew Morton Applix pty limited 2nd December 1989

cal - print calendar

cal [`month`] `year`

Description

cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. Year can be between 1 and 9999. The month is a number between 1 and 12. The calendar produced is that for England and her colonies (like the USA and Australia). Arguments are in decimal numbers, unlike 1616/OS.

If an single argument of less than 12 is given, it is considered a month.

If given two arguments, but in the wrong order (year before month) it usually gets them right. This version is smarter than your average UNIX version.

Examples

Try September 1752. (Type `cal 9 1752`) In that month, 11 days were skipped to make up for leap year adjustments not made previously (19 days hath September?)

Bugs

The year is always considered to start in January, even though this is historically naive. Beware that ‘‘cal 89’’ refers to the early Christian era, not the 20th century (it is easy to miss meetings this way!)

Arguments are considered decimal numbers, not hexadecimal, unlike 1616/OS shell. **Do not** use **.** before numbers.

If given a single argument of less than 12, a month is assumed, however this version then uses the calendar for the year 10, which isn’t all that much use.

Needs to learn how to read the Applix 1616 system date, and use the current month and year.

Does not understand month names or abbreviations, which is rather naive.

Associated files

`cal.c`

See also

date, **swget**, **swset**

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Martin Minow

Conversion to Applix 1616 by Andrew Morton

cmp - compare two files

cmp [**-ls**] `file1` `file2`

Description

The two files are compared. (If `file1` is **-**, the standard input is used.) Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted. Use it for writing quick and dirty data-bases.

-l Print the byte number (decimal) and the differing bytes (octal) for each difference (byte numbering begins at 1 rather than 0).

-s Print nothing for differing files; return codes only.

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument. Well, maybe in the UNIX version!

Examples

Bugs

Associated files

`cmp.c`

See also

comm, **diff**.

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Paul Polderman and Michael Huisjes

Conversion to Applix 1616 by Andrew Morton

comm - lines common to two files

comm [**-** [**123**]] `file1 file2`

Description

Comm reads `file1` and `file2`, which should be ordered in ASCII collating sequence (see **sort**), and produces a three column output: lines only in `file1`; lines only in `file2`; and lines in both files. The file name **-** means the standard input. This is also of use in writing quick and dirty databases.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm -12** prints only the lines common to the two files; **comm -1** prints only lines in the first file but not in the second; **comm -123** is a no-op.

Examples

Bugs

Associated files

`comm.c`

See also

cmp, diff, sort, uniq.

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Martin C Atkins

Conversion to Applix 1616 by Andrew Morton

cron - run regular background tasks

cron [-f crontabfile] [-s sleeptime] [-r rereadtime] [-v] [-d] [-e[dev-name]]

Introduction

cron is a program which is designed to run as a daemon under 1616/OS. It runs other programs at specified times and dates. cron reads a plain text file to determine what commands the user wishes to have executed and when they are to be executed.

Installation

Typically cron is started up by a command in your autoexec file at boot time. Of course cron will stop if you reset the 1616 with Alt Ctrl R or if the 1616 crashes on an exception. A modified version of the 1616 boot block program has been supplied. This version attempts to run a file called autoexec1 at level 1 and level 2 resets. This program can then restart cron. Copying a boot block, autoexec1.shell and cron.xrel to the RAM disk can speed up the reboot process, as the computer will boot from the RAM disk. This will ensure that cron is always available after a reset on a floppy drive system. The cron tables in /usr/lib/crontab must also be available.

cron's standard output and standard error

When cron runs a program that program inherits its standard input, output and error devices from cron. To prevent cron programs from scribbling on the screen at inopportune moments it is best to redirect cron's standard output and standard error to the NULL: character device driver.

Invoking cron

cron [-f crontabfile] [-s sleeptime] [-r rereadtime] [-v] [-d] [-e[dev-name]]

this program must be started asynchronously.

-f crontabfile

Specifies the file which contains the tables which tell cron what to run and when.

If you use the default name of /usr/lib/crontab you will have to set up assigns which make /usr/lib a valid directory.

-s sleeptime

Specifies the number of seconds for which cron sleeps between examining its command database and possibly the crontab file. Decreasing the sleep time will give higher resolution but will consume more CPU time.

-r rereadtime

cron periodically polls the crontab file to see if it has been altered. If it has, it is rescanned. This polling normally occurs every two minutes (120 seconds). This option allows the alteration of the reread period. It is specified in seconds. Sending cron a 1616/OS signal number 100 will also cause it to reread the crontab file. Type syscall .129 .11 cron .100 0 -v Verbose mode.

Don't use. -d Debug mode.

Don't use. -e Echo a little startup message onto standard output. Not much use if cron's standard output is redirected to NULL:

-e*devname*

Echo the startup message to the file/device '*devname*'. For example,
```
cron -eCON: >null: }null: <null: &
```
cron has a minimum resolution of 15 seconds. The most frequently that a program can be run is once every 15 seconds. The most accurately that its run time can be specified is within 15 seconds.

The crontab file format

The crontab file consists of lines which tell cron what programs to run and when to run them. Comment lines are permitted; they must start with a semicolon. Blank lines and any syntactically incorrect lines are ignored.

The line format is as follows:

Second Minute Hour Day Month Command
where the 'Second' to 'Month' fields represent a range of different times at which the command is to be run. The numeric fields consist of a single decimal number, numbers separated by commas, ranges of numbers separated by hyphens, or combinations of these. The fields are separated by spaces or tabs.

Examples: 10 12,13 10,1-5 7-9 1,2-7,4,4-9

An asterisk '*' can be used to specify all possible values of a field. The 'Day' field may also be expressed as 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat' or 'Sun'. It does not matter whether the days of week are entered in upper or lower case.

When cron detects that the current second, minute, hour, day and month all correspond to values which have been given in the crontab entry the string 'Command' has an '&' pasted onto the back of it to make it run asynchronously with respect to cron and it is passed onto the ***exec*** system call for execution.

If the 'Command' field has multiple commands separated by '!' characters then it should be surrounded by double quotes. Otherwise the program which is run could block cron.

Consider:
```
; Run modem program at midnight, delete old file
0 0 0 * * * modemthing < sa: ! cleanup
```
at midnight cron will ***exec*** the following string:
```
modemthing < sa: ! cleanup &
```
If 'modemthing' does not terminate cron will not rerun. The solution is to modify the crontab entry as follows:
```
; Run modem program at midnight, delete old file
0 0 0 * * * "modemthing < sa: ! cleanup"
```
so cron ***exec***s
```
"modemthing < sa: ! cleanup" &
```
which will immediately return control to cron. Of course the time specification fields must be within range for minutes, hours in the day, etc. If you put in a bad entry, such as 50 for the hours then the whole entry is ignored. Setting the verbose or debug flags when you start up cron will result in diagnostics being displayed for bad crontab entries. The seconds field is rounded to the lowest multiple of 15 seconds, so the seconds 0 through to 14 are are all equivalent, as are the seconds 15 through to 29, etc.

Some example crontab entries:

```
; Run every ten minutes
; sec min hour day month command
0 0,10,20,30,40,50 * * * date >> /h0/runlog

; Run at 8:00 am every day
; sec min hour day month command
0 0 8 * * echo Get up and go to work >con:
```

```
; Wish me happy birthday
; sec min hour day month command
0 0 12 15 7 echo Happy Birthday >con:

; Check the Hard disk is OK at midday
; sec min hour day month command
0 0 12 * * fscheck /h0 >/f0/fschecklog

; Beep once every 15 seconds for the first 3 minutes of
; each hour
; sec min hour day month command
0-59 0-2 * * * echo -n ^G >con:

; Remind you to buy the Herald
; sec min hour day month command
0 0 12 Mon * echo "Buy the paper" >con:

; Blow the machine away every hour, on the hour,
; but only on Fridays and the 13'th of the month
; sec min hour day month command
0 0 * 13,fri * syscall .101
```

It is not recommended that you put the last entry into your own crontab file. Put it out as shareware.

```
; Print the date every thirty seconds
0,30 0-59 0-23 0-30 0-11 date

; Really useless
0-59 * * * * echo "Hello, world"
```

'command' must be suitable for asynchronous ***exec***. `cron` sticks an '&' on the end and ***exec***s it. This means that a command like

```
fred ! bill
```

will be ***exec***ed as

```
fred ! bill &
```

so `fred` will run synchronously and block `cron`, which seems dumb. The above command should be

```
"fred ! bill"
```

so `cron` ***exec***s

```
"fred ! bill" &
```

which runs asynchronously with respect to `cron`.

Andrew Morton Applix pty limited 2nd December 1989

dd - convert, reblock, translate, and copy a (tape) file

dd `[option=value] ...`

Description

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default in the UNIX version. In some systems, the input and output block size may be specified to take advantage of raw physical I/O.

Well, I've only used **dd** to convert lower case files to upper case, and vice versa (and Andrew wrote assembler utilities to do that!). As soon as I work out what everything does, I'll write it up. Meanwhile, here are some possible options.

ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size *n* bytes (default 512)
bs=*n*	set both input and output block size, superseding **ibs=** and **obs=**.
if=`file`	input file name
of=`file`	output file name
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
files=	unknown option
length=	unknown option, block length probably
conv=lcase	map alphabetics to lower case
conv=ucase	map alphabetics to upper case
conv=swab	swap every pair of bytes
conv=noerror	do not stop processing on an error
conv=sync	pad every input block to **ibs**

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b** or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate a product.

After completion, **dd** should report the number of whole and partial input and output blocks.

Examples

Bugs

Associated files

`dd.c`

See also

tr

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Conversion to Applix 1616 by Andrew Morton

df - dump contents of a disk file

df `filename` [**-n***nnnn*] [**-o***nnnn*]

Description

Reads a specified file from disk, and displays the contents in hexadecimal and ACSII, together with the address within the file. Lazy variation on using `mload` and `mdb`.

-n*nnnn* number of bytes to read from file, in hexadecimal

-o*nnnn* offset from start of file, in hexadecimal bytes

Examples

Bugs

Only accepts a single filename, so be careful about how you use wildcards.

Associated files

`df.c`

See also

mdb and other monitor commands

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

diff - differential file comparator

diff `file1 file2`

Description

Diff tells which lines differ in two files.

Except in rare circumstances, **diff** finds a smallest sufficient set of file differences. That is, it is relatively smart about finding differences.

Examples

Bugs

Runs out of memory if there are too many differences. Truncates (ignores) input after 128 characters in long lines.

Associated files

`diff.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Erik Baalbergen, alterations by BDE.

Conversion to Applix 1616 by Andrew Morton

dis - disassembler for 68000 code

dis `start address count`
dis `filename.exec`
dis `filename.xrel`

Description

Disassembles memory starting at an address and continuing for count. Will disassemble a named file (provided the file ends with an `.exec` or `.xrel` extension.) Knows about Applix1616 syscalls, and displays an appropriate string when they are encountered.

Examples

Bugs

Associated files

`dis.c, mdbdis.c, htoc.c, main.c, sysc.c`

See also

Distribution

Applix 1616 Utility Disk #2 /disassembler/

Author

Minix disassembler for the debugger, by Bruce D Szablak.
Conversion to Applix 1616 by Andrew Morton

eroff - text justifier and formatter

eroff [*+00*] [*-00*] [**-s**] [**-h**] [`files`]

Description

A version of **roff**, extended with printer additions by Craig Mills. **Eroff** formats text contained in files (standard input by default) for printing on typewriter-like devices and line printers. The text is interspersed with lines of format control information, while the output is a printed, paginated document in a user-designated style. Device independent printing is a truly great idea, provided your particular printer is supported ... mostly, it isn't. But you have the source code, so you can change that! Craig's additions provide subscript, superscript, italics, underline, emphasised print, enlarged print, and NLQ. Very powerful, but not precisely friendly.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The options, which may appear in any order, but must appear before the files, are:

+ *n* Starting line number.

- *n* End line number.

-h Insert tabs to replace spaces wherever possible.

-s *n* Stop every *n* pages. Roff will halt after every *n* pages (default *n*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or newline. When **roff** halts between pages, an ASCII BELL is sent to the terminal.

Roff uses a large number of two character macro commands, preceded by a **.** within its text files to control formatting. An *n* indicates a number is required. A *t* indicates text is to be entered. A *c* indicates a single character is to be entered. The commands probably include:

.ad Adjust output lines.

.ar Arabic page numbers (default).

.bl *n* Blank line in text, paragraph indicator. Insert *n* blank lines.

.bp *+n* Break page, begin new page, number it *n*.

.br Break, stop filling current line. Use between paragraphs.

.cc *c* Set control character to *c*, default is **.** .

.ce *n* Center following *n* input text lines.

.ch Prompt for a printer DIP switch change from normal (0) to IBM (1).

.de *xx* Define or redefine macro *xx*, end at line beginning **..** .

.ds Double space, same as **.ls** *2*.

.ec *c* Set escape character to *c* Default is \

.ef *t* Even footer becomes *t*.

.eh *t* Even header becomes *t*.

.eo Turn escape mechanism on or off.

.fi Fill output lines, right justify margin. Default is on.

.fo *t* Footer title, default none.

.hc *c* Hyphenation indicator character *c* Initially usually none.

.he *t* Header title, default none.

.hx Title lines are suppressed, default 1.

.hy *n* Hyphenate. If *n* is 1, hyphenate, if *n* is 0, don't.

.ig .. Ignore input until you encounter ..

.in *n* Indent *n* spaces, normally *n* is zero

.ix *n* Next indent size, same as **.in**, but without break

.li *n* Literal. Treat next *n* lines as text.

.ll *n* Line length, initially 6.5 inches, 65 characters, including indent.

.ls *n* Output *n-1* vertical spaces after each line, default 1.

.m1 *n* Put *n* blank lines between top of page and header title.

.m2 *n* Put *n* blank lines between header title and start of text.

.m3 *n* Put *n* blank lines between end of text and footer.

.m4 *n* Put *n* blank lines between footer and bottom of page.

.n1 Add 5 to page offset, number lines in margin from 1 on each page.

.n2 *n* Add 5 to page offset, number lines on page starting ffrom *n*.

.na No adjusting of output line, equivalent of **.ad =0**.

.ne *n* Begin new page, if *n* lines won't fit on present page.

.nf No filling or adjusting of output lines.

.ni +*n* Line numbers are indented *n*.

.nn +*n* Number next *n* lines, initially 0.

.nx *filename* Input from next file.

.of *t* Odd footer becomes *t*.

.oh *t* Odd header becomes *t*.

.pa +*n* Same as **.bp**, start new page numbered *n*.

.pl *n* Page length, defaults to 11 inches usually, or 66 lines.

.po *n* Page offset. Preceed lines by *n* spaces.

.rm *n* Right margin setting, default 60.

.ro Roman numerals on page numbers.

.sk *n* Skip *n* blank pages.

.sp *n* Space down *n* lines (except at top of page), default 1.

.ss Single space, same as **.ls** *1*.

.ta *Nt* Tab settings, left, unless *t=R (Right)*, or *t=C (Center)*. 8 initially (positions 8, 17, 25 ...).

.tc *c* Tab replacement character, initially a space.

.ti *n* Temporary indent of *n*, for next line.

.tl *t* Print title *t*.

.tr *abcd* ... Translate *a* to *b*, etc., on output.

.ul *n* Underline letters and numbers in next *n* input lines.

\-1 Set continuous underline mode.

_1 Set non-continuous underline mode.

\I1 Set italic mode.

\E1 Set emphasised mode.

\P1 Set superscript mode.

\B1 Set subscript mode.

\W1 Set enlarged mode.

\Q1 Set letter quality mode.

\ (a space character) Make an unpaddable space sized character. The gap will not be filled with extra spaces, nor broken over a line.

\ 0 Cancel any of the above modes by repeating it, but with a **0** in place of the **1**.

Examples

Bugs

Associated files

```
eroff.c, eroff.doc, eroff.obj, eroff.xrel, expand.as, expand.obj,
makefile, runoff.s
```

See also

cat, **cio**, **more**, **pr**, **roff**, **runoff**, **dde**

Distribution

Applix 1616 Utility Disk #2 /eroff

Author

G L Sicherman, with fixes by Tim Maroney, Dave Tutelman. Refer to Kernighan & Plauger book *Software Tools* for a description.

Conversion to Applix 1616 by Andrew Morton
Printer additions from Craig Mills

exp - evaluate arguments as an expression

exp `arguments`

Description

This is usually called `expr` but that name has been used by 1616/OS. The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped using ". Note that 0 is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted.

The operators and keywords are listed below. Characters that need to be escaped must be surrounded by ".

exp | exp
: **OR** returns the first **exp** if it is neither null nor 0, otherwise returns the second **exp**.

exp & expr
: **AND** returns the first **exp** if neither **exp** is null or 0, otherwise returns 0.

exp =, >, >=, <, <=, != exp
: returns the result of an integer comparison if both arguments are integers.

( )
: Parentheses permit grouping within expressions.

exp +, - exp
: addition or subtraction of integer-valued arguments.

exp *, /, % exp
: multiplication, division, or remainder of the integer-valued arguments.

Examples

Bugs

Only 6 levels of parentheses are provided. It is very easy to forget to escape characters with ".

Associated files

`exp.c`

See also

expr

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Erik Baalbergen

Conversion to Applix 1616 by Andrew Morton

find - find a named file on disk

find `dirspec filespec`

Description

Find locates files, and reports which directory they are in. Drive is the default drive, but can be specified within the `dirspec`. Do not end the `dirspec` with a /. Wildcards are expanded correctly, however you **must** enclose then in double quotes (") to prevent them being expanded by the operating system. Output can be redirected with > for later perusal.

Examples

```
find /f0 filename
```

```
find /f1/bin "*.xrel"
```

Bugs

Specifying a root directory can be confusing. `Find /h0/ filename` is incorrect.

Associated files

`find.c`

See also

tree, **ftree**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

fscopy - copy a directory, and all subdirectories

fscopy `sourcedir destdir`

Description

Fscopy copies a directory (or a whole disk), and all subdirectories and files, to another directory or disk. Provides messages as to the directory being scanned.

You need to be careful to provide a directory name at the destination, or all subdirectories will end up in the root directory.

Examples

Bugs

Takes forever to copy large directories, so use it only to copy complex directory structures. Do not use it to copy a whole disk (it works, but is too slow). If a directory set up for more than 64 files is `fscopied`, the new directory may default to 64 files. This may cause serious disk map problems.

Associated files

`fscopy.c`

See also

copy, **diskcopy**, **fastcopy**, **tree**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

ftolower - convert input to lower case

ftolower `< infile > outfile`

Description

Converts any upper case characters in the input into lower case. Use < and > redirections.

Examples

Bugs

Associated files

`ftolower.s`

See also

addcr, **dd**, **ftoupper**, **rmcr**, tr

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

ftoupper - convert input to upper case

ftoupper `< infile  > outfile`

Description

ftoupper converts any lower case alphabetical characters to upper case. Use < and > redirections.

Examples

Bugs

Associated files

`ftolower.s`

See also

addcr, **dd**, **ftolower**, **rmcr**, tr

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

gensrec - output in S-Record form

gensrec `infile > output`

Description

Displays a specified input file in Motorola S Record format. Redirect the output as desired using >. You can waste vast amounts of time by redirecting the output into the 1616's inbuilt `srec` command.

Examples

Bugs

Associated files

`gensrec.c`

See also

sd

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

gp - fast search a file for a simple pattern

gp pattern `file`

Description

gp searches the input file for lines matching a pattern. Normally, each line found is copied to the standard output.

Examples

gp `aeiou f0/dir1/*.doc`

This finds all lines containing aeiou.

Associated files

`gp.c` There may not be an executable kicking round.

See also

find, **grep**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

grep - search a file for a pattern

grep [**-cfinv**] pattern [`file ...`]

Description

grep searches the input files (standard input is the default, so it also works as a filter) for lines matching a pattern. Normally, each line found is copied to the standard output. **Grep** patterns are limited "regular expressions" in the style of UNIX ed.

I'll eventually add a detailed description of how to form a regular expression. Meanwhile, you can use **grep** to search for strings (just enclose them in quotes ") without much effort, but it is actually far more powerful than any usual word processor search function. Use it also for quick and dirty databases.

Here are the command line options, used to include or exclude specific lines of a file in the output.

-c Only a count of matching lines is printed.

-f File switch. Normally, file names are only printed if more than one file is searched. Switch makes file name print if there is one file only.

-i Ignore upper/lower case distinction during comparisons (normally case sensitive).

-n Each line is preceded by its relative line number in the file.

-v All lines **except** those matching are printed.

Care should be taken when using the characters $, *, ^, |, etc., in the expression of a pattern, because they are also meaningful to the shell. It is safest to enclose the entire pattern argument in quotes. Patterns must be quoted if they contain spaces. Wildcards and pathnames are allowed in the file specification.

Within a regular expression, you can 'anchor' the search pattern to the start of a line using ^, or the end of a line using **$**. As well as matching any string of characters, you can look for an arbitrary number of occurences of a character by using ***** or **+**. You accept any arbitrary character by using **.** to stand for any number of characters.

By using the **[ ]** surrounding characters, you can search for *any* of the bracketed characters at that position. Including ^ as the first character within the bracket means to exclude the other characters within it. Pattern matching understands ASCII sequence, so you can group characters using **-**, so A-N means the range of characters from A up to N.

Characters that are available for use in forming regular expressions include:

***** Zero or more occurrences of the previous character.

+ One or more occurrences of the previous character.

- Optionally match previous character (say what?)

**** Escapes any special character.

^ Starts the pattern at the beginning of a new line.

$ Indicates the end of a line.

. Matches any character except end of line.

:a alphabetical characters

:d digits

:n non alpha characters

‘**:** ’ whitespace characters, including tabs (this is a colon followed by a space)

[ ] Matches a class of characters.
Matches any character in a set. Ranges such as [a-z] or [1-9] are allowed. Thus [aeiou] matches any vowel. ^ as the first character in a class means match anything except what is in the class, so [^aeiou] would match any consonant.

Examples

```
grep -fi "[aeiou]+: " f0/dir1/*.doc f1/tutor/*.text
```

This finds all vowels that are followed by a space in the two files named. More prosaic uses include simply searching files for a given string, as in `grep "string" file1 file2`.

Associated files

`grep.c`

See also

find, **gp**

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

DECUS C Tools, modified by Chuck Allison

Conversion to Applix 1616 by Andrew Morton

hdbackup - backup a hard disk to floppies

hdbackup `/hdrive /fdrive`

Description

Hard disk backup utility, intended to output to 800k floppies only. Reports source and destination, buffer size, blocks used, and number of floppies required. Internally numbers the floppies that it uses (so label them correctly!) Prompts for new floppies as required.

Examples

Bugs

On disk error produces `Retry, Ignore, Abort` messages (did you *really have to imitate MS-DOS*, Andrew?) Only knows about 800k floppies, so don't try backing one hard disk to another!

Associated files

`hdbackup.c`

See also

copy, **hdrestore**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

hdrestore - restore contents of a hard disk

hdrestore `/sourcedrive /destdrive`

Description

Restores a hard disk that has been saved on floppies using `hdbackup`. Warns that this will destroy the contents of the hard disk (so get the partition right!), and prompts for next floppy (so give it the disks in the correct order!)

Examples

Bugs

Associated files

`hdrestore.c`

See also

copy, **hdbackup**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

head - display top of file

head [**-n**] `file1 [file2 ... ]`

Description

Display the first *n* lines in a file or group of files. The default is 10 lines. Used to quickly survey the contents of files.

-n Number of lines to be displayed.

Examples

Bugs

Associated files

`head.c`

See also

tail

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Chuck Allison

Converted to Applix 1616 by Andrew Morton

load4000 - convert .xrel to .exec

```
load4000 filename
```

Description

`load4000 filename` loads a named `.xrel` file into memory at $4000. Use the inbuilt `msave` command to write it back out as an `.exec` file.

Examples

Bugs

Associated files

`load4000.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

make - maintain update and regenerate groups of programs

make [**-f** `makefile`] [**-dinpqrst**] [`macro=val`] [`target(s)`] [`names`]

Description

Make automates the updating of repeated compilations, where only some modules of a large program are altered. It is not required when a C program consists of only a single source code file.

In a large C program, consisting of several source code files, header files, and libraries, modifying any one of these means you will have to recompile. To recompile all modules, and relink everything to produce a new executable file, is tedious and time-consuming. `Make` allows you to recompile only the changed files, **and** those files that depend upon the chnged file contents, and relink these new object modules with those from a previous link.

`Make` takes a file of source-code dependencies (called the `makefile`), and automatically performs all necessary compiles and links, to create the final target file, the executable program. It compares the time and date stamp on files to determine which files are newly altered (you can force inclusion of a file by using `touch` to update its time and date stamp). `Make` uses the details in the `makefile` (which the author of the program writes initially) to select the files which are dependent upon newly altered files. You will find working `makefiles` in some of the Applix 1616 shareware disks (look at the `makefile` for `make`, for instance). `Makefiles` often contain specifications for the libraries to be used, compiler directives, clean up operations and so on. `Make` itself also includes internal rules to assist in working out dependencies, and these rules are always overridden by the contents of a `makefile`.

Typically, the `makefile` lists a target files, and then lists the files upon which it depends. It typically also includes a list of actions to be taken if the target file is to be remade.

The following is a brief description of all options and some special names. Options can occur in any order.

-d debug output to go to stderr. Print out detailed information on files and times examined. (This is intended for debugging the make command itself.)

-f makefile Description file name. Makefile is assumed to be the name of a description file. A file name of **-** denotes the standard input. The contents of `makefile` override the built-in rules, if they are present. Note that the space between **-f** and `makefile` must be present.

-i Ignore error codes returned by invoked commands. This mode is also entered if the fake target name .IGNORE appears in the description file.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

-q Question. The make command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

-p Print out the complete set of macro definitions and target descriptions.

-r Do not use the built-in rules.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

The "built-in" dependency targets are:

.PRECIOUS

Dependents of this target will not be removed when QUIT or INTERRUPT are hit.

.SILENT

Same effect as the -s option.

.IGNORE

Same effect as the -i option.

Make executes commands in `makefile` to update one or more target `names`. `Name` is typically a program. If no **-f** option is present, `makefile`, and `s.makefile` are tried in order. If `makefile` is **-**, the standard input is taken. More than one **-f** `makefile` argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

`Makefile` contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, followed by a colon (:), followed by a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab (spaces are not an acceptable substitute, although they probably should be) are shell commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is.

Sharp (#) and new-line surround comments before the rules.

The following `makefile` says that `pgm` depends on two files `a.o` and `b.o`, and that they in turn depend on their corresponding source files (`a.c` and `b.c`) and a common file `incl.h`:

```
pgm: a.o b.o
     cc a.o b.o -o pgm
a.o: incl.h a.c
     cc -c a.c
b.o: incl.h b.c
     cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: -, @, -@, or @-. If @ is present, printing of the command is suppressed. If - is present, `make` ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry .SILENT: is in makefile, or unless the initial character sequence contains a @.

The **-n** option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed. Note that this feature does not work if MAKE is enclosed in braces, as in ${MAKE}. The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate make. If the **-i** option is present, or the entry .IGNORE: appears in makefile, or the initial character sequence of the command contains -. the error is ignored.

INTERRUPT and QUIT cause the target to be deleted unless the target depends on the special name .PRECIOUS.

Macros

Entries of the form string1 - string2 are macro definitions. String2 is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(string1[:subst1= [subst2]]) are replaced by string2. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :subst1=subst2 is a substitute sequence. If it is specified, all non-overlapping occurrences of subst1 in the named macro are replaced by subst2. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines.

Internal Macros

There are four internally maintained macros that are useful for writing rules for building targets.

$* The macro $* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@ The $@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$< The $< macro is only evaluated for inference rules or the .DEFAULT rule. It is the module that is out-of-date with respect to the target, i.e., the ‘‘manufactured’’ dependent file name. Thus, in the .c.o rule, the $< macro would evaluate to the .c file. An example for making optimized .o files from .c files is:

```
.c.o:
     cc -c -O $*.c
```

or:

```
.c.o:
     cc -c -O $<
```

$? The $? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules that must be rebuilt.

Suffixes

Certain names (for instance, those ending with .o) have inferable prerequisites such as .c, .s, etc. If no update commands for such a file appear in `makefile`, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, **make** has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

This is because `make` has a set of internal rules for building files. The user may add rules to this list by simply putting them in the `makefile`.

The inference of prerequisites can be controlled. The rule to create a file with suffix `.o` from a file with suffix `.c` is specified as an entry with .c .o: as the target and no dependents. Shell commands associated with the target define the rule for making a `.o` file from a `.c` file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

WARNINGS

Be wary of any file (such as an include file) whose access, modification, and last change times cannot be altered by the **make**-ing process. For example, if a program depends on an include file that in turn depends on another include file, and if one or both of these files are out-of-date, `make` will try to update these files each time it is run, thus unnecessarily re-make-ing up-to-date files dependent on the include file. The solution is to manually update these files with the `touch` command before running **make**.

(Note that it is generally a bad idea to include the `touch` command in your `makefile`, because it can cause **make** to update a program that otherwise did not need to be updated.)

Check `rules.c` to find changes to defaults. For example, the assembler AS is now `as68k`, ASFLAGS is `-q`, CC is now `relcc`, and so on, as required for use with Applix 1616 modified versions of HiTech C.

Examples

Bugs

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

File names with the characters - : @ will not work.

Make will not properly expand a macro within another macro when string substitution is involved.

Associated files

```
00readme, 00revhst.txt, check.c, check.obj, errs, h.h, input.c,
input.obj, macro.c, macro.obj, main.c, main.obj, make.c, make.help,
make.obj, make.xrel, makefile, reader.c, reader.obj, rules.c,
rules.obj, system.c, vaxvms.c
```

See also

relcc, **easy_write**

Distribution

Applix 1616 Utility Disk #2 /make

Author

Conversion to Applix 1616 by Andrew Morton

mexec - multiple exec

```
mexec [-v] printf_control_string [args]
```

Description

`mexec [-v] printf_control_string [args]` executes ONE 1616/OS command ONCE upon each of its arguments. If the `-v` flag is given, `mexec` echos the commands it is about to ***exec*** before doing it.

Examples

For example, `mexec "delete %s" *` is a complicated way of deleting every file in the current directory. In this, `mexec` takes every argument, replaces every occurrence of `%s` in the command string with the argument, and ***exec***s the result.

Another example, `mexec -v "rename %s %s.zz" *.s` adds a `.zz` to the end of every `.s` file in the current directory.

A more complicated example, `mexec "rmcr %s ! addcr %s ! edit %s" *.c`, will go through every `.c` file in the directory, run them through `rmcr` and `addcr` before editing them.

You could use `mexec "edit %s" file1 file2 file3 *.doc` to sequentially edit a bunch of files.

The `mexec` program is designed so that if NO arguments are given after the ***printf*** control string, the arguments are read from standard input until it encounters an end-of-file. This means that the output from the `find.xrel` program can be directed into the input of `mexec` to provide it with its list of filenames. This may be done using `pipe` or by going through a temporary file. For example

```
cd /f0
find . "*.bak" | mexec -v "delete" %s
```

will delete all the editor backup files from all directories in `/F0`. The same effect can be obtained via temporary files, as below.

```
cd /f0
find . "*.bak" > /rd/fred
mexec - v "delete %s" < /rd/fred
delete /rd/fred
```

Bugs

Associated files

`mexec.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

more - formats text files for viewing

more `[ files ... ]`

Description

More is a filter which allows examination of continuous text, one screenful at a time, on a display. It knows about wildcards and pathnames. It is retained in UNIX for backward compatibility (stuff like `pg` or `less` are used instead). As with all filters, you can feed it from standard input, and get results from standard output, rather than using filenames.

It pauses after each screenful, printing -More- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, or almost any other character, another screenful is displayed. Use it instead of `type` or `edit` for reading a file. Other characters used interactively include:

n Next file is brought in for viewing.

N Next file is brought in for viewing.

Q Exit from **more**.

q exit from **more** (same as Q).

CR Display one more line.

Examples

A sample usage of **more** (in UNIX) in previewing **roff** output would be

```
roff -s +2 doc.n | more
```

Bugs

When first invoked, the first line of the file is lost off the top of the display.

More does not scroll backwards.

Associated files

`more.c`

See also

cat, **cio**, **pr**, **roff**, **more.s** (better version from Matthew Gardner)

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Chuck Allison

Conversion to Applix 1616 by Andrew Morton

mrdstat - reports on mrdrivers in memory

`mrdstat`

Description

Dumps out information on the names and locations of any memory resident drivers (MRDs) loaded by the `mrdrivers` on the disk from which you booted.

Examples

Bugs

Associated files

? Don’t know where source is for this.

See also

Distribution

Applix 1616 Utility Disk #1 /bin

Author

Andrew Morton

pipe - adds UNIX pipes as MRD

^ or |
pipe `[ on ] [ off ] [ d ]`

Description

Pipe is a memory resident driver, to be added to your `mrdrivers` file on your boot disk for Version 3. Refer to the *Applix Technical Reference Manual*. Pipes are built into the operating system from Version 4.

It is invoked using ^ for standard output, or | for standard error. It acts as a redirection operator, taking the output of the file preceding it, and using this as the input of the file following it. It does this by producing temporary disk files (like MS-DOS, unlike UNIX.)

UNIX folks should note that the ^ character is historically correct. Since standard error is normally directed to standard output, you can use the more familiar | in most circumstances without any different effect. Most versions floating round have already been converted so that | is the usual default for standard output (so much for history!) The pipe in Version 4 uses only |.

on Enable piping

off Disable the **pipe** mrd

d Debug mode. Lists all expansions made by **pipe**.

Available within 1616/OS from Version 4 on.

Examples

```
cat filename ^ wc
```

produces a count of the number of lines, words and bytes in `filename`.

```
dir | more
```

Displays a directory a page at a time, via the `more` filter.

Bugs

Associated files

`pipe.c, pipe.mrd`

See also

Distribution

Applix 1616 Utility Disk #1 /source/mrdrivers

Author

Andrew Morton

pr - print files

pr [+*page*] [-*columns*] [**-h** `header`] [**-w** *width*] [**-l** *length*] [**-bfnpst**] [`files`]

Description

Pr prints the named files to the standard output. If file is **-**, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file. Redirect this to your printer for a formatted printout.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the | character.

The options below may appear singly or be combined in any order:

+ Page number. Begin printing with page *n* (default is 1).

- Number of columns. Produce *n*-column output (default is 1, limit is 10).

-b Backspace correction turned off.

-f Folding of lines to fit the display is to be disabled.

-h Use the next argument as the header to be printed instead of the file name. **-h is the only pr option that requires a space between the option and its argument!**

-l Length. Set the length of a page to *n* lines (default is 66). By default, pages contain

-n Line number. Provide *n*-digit line numbering (default for *n* is 5). The number occupies the first *n+1* character positions of each column of normal output.

-p Pause before beginning each page if the output is directed to a terminal (**pr** will ring the bell at the terminal and wait for a carriage return).

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

-w Width. Set the width of a line to *n* character positions (default is 72 for equal-width multi-column output).

Examples

Print `file1` and `file2` as a three-column listing headed by ‘‘filelist’’:

```
pr -3h "file list" file1 file2
```

Write `file1` on `file2`, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -t <file1 >file2
```

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Michael Huisjes, modified by Jacob P Bunschoten.

Conversion to Applix 1616 by Andrew Morton

rawread - read a block device

rawread `/device address blocknum [ blocknum ]`

Description

Bypasses the operating system, and calls a block device driver directly to read disks. This is identical to **rawwrite**, and looks at its own call name to determine which action is required (cute trick). Use to rescue destroyed disks.

/device is the disk drive (/f0, /h0, etc.)

address is the location in memory into which the results are to be placed. The value is assumed to be hexadecimal. Using $6000 to $8000 is often safe.

block The number of the block, or blocks, to be read.

Examples

```
rawread /f0 6000 3 4
```

should read the directory blocks (which are usually blocks 3 and 4) of the disk in drive /f0, and place the contents in memory at $6000. You can check the results using **mrb** 6000.

Bugs

Does not accept decimal or binary numbers for address.

Must not be renamed, or it stops! Actually, this could be considered a feature!

Associated files

`rawread.c`

See also

rawwrite

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

rawwrite - write to a block device

rawwrite `/device address blocknum [ blocknum ]`

Description

Bypasses the operating system, and calls a block device driver directly to write disks. This is identical to **rawread**, and looks at its own call name to determine which action is required (cute trick).

/device is the disk drive (/f0, /h0, etc.)

address is the location in memory from which the contents are to be taken. It normally uses 1k per block. The value is assumed to be hexadecimal. Using $6000 to $8000 is often safe.

block The number of the block, or blocks, to be written.

Examples

```
rawwrite /f0 6000 3
```

should write to the directory block (which is usually block 3) of the disk in drive /f0, and place the contents of memory from $6000 in that block. You can alter memory using **mwb**.

Bugs

Does not accept decimal or binary numbers for address.

Can not be renamed, or it stops. Actually, this could be considered a feature!

Associated files

`rawwrite.c`

See also

rawread

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

rmcr - remove carriage returns from file

rmcr `infile outfile`

Description

Removes carriage returns from ends of lines in file `infile`, and places the result in `outfile`. Use to help fix files with strange CR-LF combinations, by running through `rmcr`, then through `addcr`.

Examples

Bugs

Requires two filenames, rather than accepting redirections.

Associated files

`rmcr.c`

See also

addcr, **ftolower**, **ftoupper**

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

roff - text justifier and formatter

roff [*+00*] [*-00*] [**-s**] [**-h**] [`files`]

Description

Roff formats text contained in files (standard input by default) for printing on typewriter-like devices and line printers. The text is interspersed with lines of format control information, while the output is a printed, paginated document in a user-designated style. Device independent printing is a truly great idea, provided your particular printer is supported ... mostly, it isn't. But you have the source code, so you can change that! Very powerful, but not precisely friendly.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The options, which may appear in any order, but must appear before the files, are:

+ *n* Starting line number.

- *n* End line number.

-h Insert tabs to replace spaces wherever possible.

-s *n* Stop every *n* pages. Roff will halt after every *n* pages (default *n*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or newline. When **roff** halts between pages, an ASCII BELL is sent to the terminal.

Roff uses a large number of two character macro commands, preceded by a **.** within its text files to control formatting. An *n* indicates a number is required. A *t* indicates text is to be entered. A *c* indicates a single character is to be entered. The commands probably include:

.ad Adjust output lines.

.ar Arabic page numbers (default).

.bl *n* Blank line in text, paragraph indicator. Insert *n* blank lines.

.bp *+n* Break page, begin new page, number it *n*.

.br Break, stop filling current line. Use between paragraphs.

.cc *c* Set control character to *c*, default is **.** .

.ce *n* Center following *n* input text lines.

.de *xx* Define or redefine macro *xx*, end at line beginning **..** .

.ds Double space, same as **.ls** *2*.

.ef *t* Even footer becomes *t*.

.eh *t* Even header becomes *t*.

.fi Fill output lines, right justify margin. Default is on.

.fo *t* Footer title, default none.

.hc *c* Hyphenation indicator character *c* Initially usually none.

.he *t* Header title, default none.

.hx Title lines are suppressed, default 1.

.hy *n* Hyphenate. If *n* is 1, hyphenate, if *n* is 0, don't.

.ig .. Ignore input until you encounter ..

.in *n* Indent *n* spaces, normally *n* is zero

.ix *n* Next indent size, same as **.in**, but without break

.li *n* Literal. Treat next *n* lines as text.

.ll *n* Line length, initially 6.5 inches, 65 characters, including indent.

.ls *n* Output *n-1* vertical spaces after each line, default 1.

.m1 *n* Put *n* blank lines between top of page and header title.

.m2 *n* Put *n* blank lines between header title and start of text.

.m3 *n* Put *n* blank lines between end of text and footer.

.m4 *n* Put *n* blank lines between footer and bottom of page.

.n1 Add 5 to page offset, number lines in margin from 1 on each page.

.n2 *n* Add 5 to page offset, number lines on page starting ffrom *n*.

.na No adjusting of output line, equivalent of **.ad =0**.

.ne *n* Begin new page, if *n* lines won't fit on present page.

.nf No filling or adjusting of output lines.

.ni +*n* Line numbers are indented *n*.

.nn +*n* Number next *n* lines, initially 0.

.nx *filename* Input from next file.

.of *t* Odd footer becomes *t*.

.oh *t* Odd header becomes *t*.

.pa +*n* Same as **.bp**, start new page numbered *n*.

.pl *n* Page length, defaults to 11 inches usually, or 66 lines.

.po *n* Page offset. Preceed lines by *n* spaces.

.rm *n* Right margin setting, default 60.

.ro Roman numerals on page numbers.

.sk *n* Skip *n* blank pages.

.sp *n* Space down *n* lines (except at top of page), default 1.

.ss Single space, same as **.ls** *1*.

.ta *Nt* Tab settings, left, unless *t=R (Right)*, or *t=C (Center)*. 8 initially (positions 8, 17, 25 ...).

.tc *c* Tab replacement character, initially a space.

.ti *n* Temporary indent of *n*, for next line.

.tl *t* Print title *t*.

.tr *abcd* ... Translate *a* to *b*, etc., on output.

.ul *n* Underline letters and numbers in next *n* input lines.

Examples

Bugs

Associated files

`roff.c`

See also

cat, **cio**, **more**, **pr**

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

G L Sicherman, with fixes by Tim Maroney, Dave Tutelman. Refer to Kernighan & Plauger book *Software Tools* for a description.

Conversion to Applix 1616 by Andrew Morton

scc - Z8530 serial preload constant calculator

`scc`

Description

`scc` is a utility for working out the preload value for the Z80 SCC (Z8530) serial I/O chip preload constant (wasn't that informative - look in the SCC manual).

Examples

Bugs

Associated files

`scc.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

setload - change load address of files

```
setload filename address
```

Description

`setload filename address` changes the load address field in a file directory entry. The address must be in hexadecimal.

Examples

Bugs

Associated files

`setload.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

sort - sort a file

sort [**-funbirdcmt**`'x'`] [+*pos* [*m.n*] [-*pos*]] [**-o** `outfile`] [`files`]

Description

Sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if **-** is used as a file name or no input files are named. Sort helps you write quick and dirty database programs.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence (ASCII order).

The following options alter the default behaviour:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only, the input files are already sorted.

-u Unique: suppress all but one in each set of lines having equal keys.

-o *output* The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between -o and *output*.

The following options override the default ordering rules.

-d Dictionary order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

-f Fold lowercase letters into uppercase.

-i Ignore characters outside the ASCII range 32-126 in non-numeric comparisons.

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first **blank** (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *x x* delimits an empty field).

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **-b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

Pos1 and *Pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n+1*st character in the *m+1*st field. A missing *.n* means .0, indicating the first character of the *m+1*st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m+1*st field; *+m.0***b** refers to the first non-blank character in the *m+1*st field.

A last position specified by *-m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m+1*st field; *-m.1***b** refers to the first non-blank in the *m+1*st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

Examples

Sort the contents of infile with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of infile1 and infile2, placing the output in outfile and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of infile1 and infile 2 using the first non-blank character of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print a file sorted by the numeric third colon-separated field:

```
sort -t: +2n -3 infile
```

Print the lines of the already sorted file `infile`, suppressing all but the first occurrence of lines having the same third field (the options -um with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

Bugs

The **-d** option recognizes ASCII characters only.

The **-n** option only recognizes the English radix character (decimal point) in numeric comparisons.

The **-t** option only recognizes a character encoded in one byte as a field separator character. That is, if you got fancy in defining your fields, it won't work.

Only handles 20k files. Change line 45 (where it says .. (20 * 1024)), compile it again, and use `chmem` to increase the stack space (if required).

Associated files

`sort.c`

See also

comm, join, **uniq**

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Michael Huisjes

Conversion to Applix 1616 by Andrew Morton

split - split a file into pieces

split [**-n**] [`file` [`name`]]

Description

Split reads `file` and writes it in *n* line pieces (default 1000 lines) onto a set of output files. The name of the first output file is `name` with `aa` appended, and so on lexicographically, up to `zz` (a maximum of 676 files, except I have a suspicion that the real maximum is 10 files). `Name` probably cannot be longer than 27 characters. If no output name is given, `xxxx` is default.

This splits a file horizontally, and can be of use for limiting file sizes, and for quick and dirty databases. To split a file vertically, use **cut** (which I don't think has been implemented as yet).

If no input file is given, or if - is given instead, then the standard input file is used.

Examples

Bugs

Associated files

`split.c`

See also

cut

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Michael Huisjes

Conversion to Applix 1616 by Andrew Morton

strings - searches for ASCII in files

```
strings [-minsize] [files]
```

Description

`strings [-minsize] [files]` searches files for ASCII strings. If no files are given, `strings` reads from standard input until it finds an EOF character. Normally it prints out any sequential run of 5 or more sequentially valid ASCII characters. The minimum run size of ASCII characters may be varied, to look for a longer or shorter run of ASCII.

Examples

Bugs

Associated files

`strings.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

sum - checksum and block count a file

sum `file`

Description

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. Standard input is used if no file names are given.

Sum is typically used to look for bad spots, or to validate a file communicated over some transmission line.

Examples

Bugs

Associated files

`sum.c`

See also

wc

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Martin C Atkins

Conversion to Applix 1616 by Andrew Morton

tail - deliver last part of a file

tail [-*number*] [`file`]

Description

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used. That is, it lets you easily inspect the end of a file.

Copying begins at distance **-***number* from the end of the input (if number is null, the value -10 is assumed). Number is counted in units of lines.

Examples

Bugs

`Tails` relative to the end of the file are stored in a buffer, and thus are limited in length. Thus, be wary of the results when piping output from other commands into `tail`.

Various kinds of anomalous behaviour may happen with character special files.

`Tail` can pick up a maximum of 4K bytes of data from the specified file.

Associated files

`tail.c`

See also

head

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Conversion to Applix 1616 by Andrew Morton

tee - pipe fitting

tee [**-i**] [**-a**] [`file`] ...

Description

Tee transcribes the standard input to the standard output and makes copies in the files. This is handy when you want to view the results of some activity, but also retain a copy of the results for later editing or printing.

-i ignores interrupts (on the Applix, who knows?)

-a causes the output to be appended to the files rather than overwriting them.

Examples

```
dir | tee -a filename
```

This displays the directory, and also puts a copy of the display into the file `filename`.

Bugs

Associated files

`tee.c`

See also

pipe.mrd

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Paul Polderman

Conversion to Applix 1616 by Andrew Morton

undelete - recover a recently deleted file

undelete [`files ...`]

Description

With no arguments, **undelete** prints a list of deleted files in the specified directory. If given a filename, or filenames, it recovers those files.

You **must** run **fscheck** after recovering a file, to correct the disk bitmap (Caution: on large hard disks, **fscheck** takes forever, and may crash without warning.)

Examples

Bugs

You may have problems with **fscheck** on large hard disks afterwards.

Associated files

`undelete.c`

See also

find, **fscheck**, tree

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Andrew Morton

uniq - compact repeated lines

uniq [**-udc** [+*n*] [-*n*]] [`input` [`output`]]

Description

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Input and output should always be different. Note that repeated lines must be adjacent in order to be found. Use **sort** to ensure that lines that are the same end up adjacent to each other. This is handy in quick and dirty databases, where you want to eliminate multiple entries with the same information. It is a pity most mail order places haven't discovered the equivalent routine in their databases. Flags are usually mutually exclusive, as detailed below.

-u just the lines that are not repeated in the original file are output.

-d specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

-c supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

-n skips *n* initial fields of each line in the comparison. Each field is a string of non-space, non-tab characters, separated by tabs and spaces from the next field.

+n The first *n* fields, together with any blanks before each, are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbours.

Examples

Bugs

Associated files

`uniq.c`

See also

comm, cut, **diff**, **sort**

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

John Woods

Conversion to Applix 1616 by Andrew Morton

wc - word, line, and character count

wc [**-lwc**] [`files`]

Description

Wc counts lines, words, and characters in the named files, or in the standard input if no names appear. It also keeps a total count for all named files. A word is a maximal string of characters (probably maximum of 256) delimited by spaces, tabs, or new-lines. Handy if your word processor can't count!

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

When `names` are specified on the command line, they will be printed along with the counts.

Wc counts the number of new-lines to determine the line count. If an ASCII text file has a final line that is not terminated with a new-line character, the count will be off by one.

If there are very many characters, words, and/or lines in an input file, the output may be hard to read. This is because **wc** reserves a fixed column width for each count. This probably doesn't matter on this version.

Examples

Bugs

Associated files

`wc.c`

See also

Distribution

Applix 1616 Utility Disk #1 /source/unix

Author

Conversion to Applix 1616 by Andrew Morton

Description

Examples

Bugs

Associated files

.c

See also

Distribution

Applix 1616 Utility Disk #1 /source/sstools

Author

Conversion to Applix 1616 by Andrew Morton

Summary

addcr `infile outfile`

ar [**-adprtvx**] `afile` [`filename`] **...**

ar [**-adprtvx**] `afile` [`filename`] **...**

arc [**-**]{`amufdxeplvtc`} [`biswn`] [g*password*] *archive* [*filename ...*]

at `HHMM [command] or`
at `MMDDHHMM [command]`

atrun `[-l[logdirectory]] filenames`

cal [`month`] `year`

cmp [**-ls**] `file1 file2`

comm [**-** [**123**]] `file1 file2`

cron `[-f crontabfile] [-s sleeptime] [-r rereadtime] [-v] [-d] [-e[dev-name]]`

dd `[option=value] ...`

df `filename` [**-n***nnnn*] [**-o***nnnn*]

diff `file1 file2`

dis `start address count`
dis `filename.exec`
dis `filename.xrel`

eroff [+*00*] [-*00*] [**-s**] [**-h**] [`files`]

exp `arguments`

find `dirspec filespec`

fscopy `sourcedir destdir`

ftolower `< infile > outfile`

ftoupper `< infile  > outfile`

gensrec `infile > output`

gp pattern `file`

grep [**-cfinv**] pattern [`file ...`]

hdbackup `/hdrive /fdrive`

hdrestore `/sourcedrive /destdrive`

head [**-n**] `file1 [file2 ... ]`

`load4000 filename`

make [**-f** `makefile`] [**-dinpqrst**] [`macro=val`] [`target(s)`] [`names`]

`mexec [-v] printf_control_string [args]`

more `[ files ... ]`

`mrdstat`

^ or |
pipe `[ on ] [ off ] [ d ]`

pr [+*page*] [-*columns*] [**-h** `header`] [**-w** *width*] [**-l** *length*] [**-bfnpst**] [`files`]

rawread `/device address blocknum [ blocknum ]`

rawwrite `/device address blocknum [ blocknum ]`

rmcr `infile outfile`

roff [+*00*] [-*00*] [**-s**] [**-h**] [`files`]

`scc`

`setload filename address`

sort [**-funbirdcmt**`'x'`] [+*pos* [*m.n*] [-*pos*]] [**-o** `outfile`] [`files`]

split [**-n**] [`file` [`name`]]

strings `[-minsize] [files]`

sum `file`

tail [-*number*] [`file`]

tee [**-i**] [**-a**] [`file`] ...

undelete [`files ...`]

uniq [**-udc** [+*n*] [-*n*]] [`input` [`output`]]

wc [**-lwc**] [`files`]

Table of Contents