LynxOS Reference Manual

LynxOS Version 2.5
DOC-0169

Utility Programs (G–O)

Printed in the United States of America.

Contents

Utility Programs (G-O)

Utility Programs (G-O)

NAME

`g++ –` GNU project C++ Compiler

SYNOPSIS

g++ [*option* **|** *filename* **] . . .**

DESCRIPTION

The C and C++ compilers are integrated; `g++` is a script to call `gcc` with options to recognize C++. `gcc` processes input files through one or more of four stages: preprocessing, compilation, assembly, and linking. This man page contains full descriptions for *only* C++ specific aspects of the compiler, though it also contains summaries of some general-purpose options. For a fuller explanation of the compiler, see `gcc`(1).

C++ source files use one of the suffixes '`.C`', '`.cc`', '`.cxx`', '`.cpp`', or '`.c++`'; preprocessed C++ files use the suffix '`.ii`'.

OPTIONS

There are many command-line options, including options to control details of optimization, warnings, and code generation, which are common to both `gcc` and `g++`. For full information on all options, see `gcc`(1).

Options must be separate: '**–dr**' is quite different from '**–d –r** '.

Most '**–f**' and '**–W**' options have two contrary forms: **–f***name* and **–fno–***name* (or **–W***name* and **–Wno–***name*). Only the non-default forms are shown here.

–c Compile or assemble the source files, but do not link. The compiler output is an object file corresponding to each source file.

–D*macro* Define macro *macro* with the string '`1`' as its definition.

–D*macro*=*defn* Define macro *macro* as *defn*.

–E Stop after the preprocessing stage; do not run the compiler proper. The output is preprocessed source code, which is sent to the standard output.

-fall-virtual Treat all possible member functions as virtual, implicitly. All member functions (except for constructor functions and `new` or `delete` member operators) are treated as virtual functions of the class where they appear.

This does not mean that all calls to these member functions will be made through the internal table of virtual functions. Under some circumstances, the compiler can determine that a call to a given virtual function can be made directly; in these cases the calls are direct in any case.

-fdollars-in-identifiers

Permit the use of '`$`' in identifiers. Traditional C allowed the character '`$`' to form part of identifiers; by default, GNU C also allows this. However, ANSI C forbids '`$`' in identifiers, and GNU C++ also forbids it by default on most platforms (though on some platforms it's enabled by default for GNU C++ as well).

-felide-constructors

Use this option to instruct the compiler to be smarter about when it can elide constructors. Without this flag, GNU C++ and `cfront` both generate effectively the same code for:

```
A foo ();
A x (foo ());
    // x initialized by 'foo ()', no ctor
      called
A y = foo ();
    // call to 'foo ()' heads to temporary,
    // y is initialized from the temporary.
```

Note the difference! With this flag, GNU C++ initializes '`y`' directly from the call to `foo()` without going through a temporary.

-fenum-int-equiv

Normally GNU C++ allows conversion of `enum` to `int`, but not the other way around. Use this option if you want GNU C++ to allow conversion of `int` to `enum` as well.

–fexternal–templates

Produce smaller code for template declarations, by generating only a single copy of each template function where it is defined. To use this option successfully, you must also mark all files that use templates with either '`#pragma implementation`' (the definition) or '`#pragma interface`' (declarations).

When your code is compiled with '**`–fexternal–templates`**', all template instantiations are external. You must arrange for all necessary instantiations to appear in the implementation file; you can do this with a `typedef` that references each instantiation needed. Conversely, when you compile using the default option '**`–fno–external–templates`**', all template instantiations are explicitly internal.

–fno–gnu–linker

Do not output global initializations (such as C++ constructors and destructors) in the form used by the GNU linker (on systems where the GNU linker is the standard method of handling them). Use this option when you want to use a non-GNU linker, which also requires using the `collect2` program to make sure the system linker includes constructors and destructors. (`collect2` is included in the GNU CC distribution.) For systems which *must* use `collect2`, the compiler driver `gcc` is configured to do this automatically.

–fmemoize–lookups
–fsave–memoized

These flags are used to get the compiler to compile programs faster using heuristics. They are not on by default since they are only effective about half the time. The other half of the time programs compile more slowly (and take more memory).

The first time the compiler must build a call to a member function (or reference to a data member), it must (1) determine whether the class implements member functions of that name; (2) resolve which member function to call (which involves figuring out what sorts of type conversions need to be made); and (3) check the visibility of the member function to the caller. All of this adds up to slower compilation. Normally, the second time a call is made to that member function (or reference to that data member), it must go through the same lengthy process again. This means that code like this

```
cout << "This " << p << " has " << n << " legs.\n"
;
```

makes six passes through all three steps. By using a software cache, a "hit" significantly reduces this cost. Unfortunately, using the cache introduces another layer of mechanisms which must be implemented, and so incurs its own overhead. '**-fmemoize-lookups**' enables the software cache.

Because access privileges (visibility) to members and member functions may differ from one function context to the next, `g++` may need to flush the cache. With the '**-fmemoize-lookups**' flag, the cache is flushed after every function that is compiled. The '**-fsave-memoized**' flag enables the same software cache, but when the compiler determines that the context of the last function compiled would yield the same access privileges of the next function to compile, it preserves the cache. This is most helpful when defining many member functions for the same class: with the exception of member functions which are friends of other classes, each member function has exactly the same access privileges as every other, and the cache need not be flushed.

-fno-default-inline

Do not make member functions inline by default merely because they are defined inside the class scope. Otherwise, when you specify **-O**, member functions defined inside class scope are compiled inline by default; i.e., you don't need to add '`inline`' in front of the member function name.

-fno-strict-prototype

Consider the declaration `int foo();`. In C++, this means that the function `foo` takes no arguments. In ANSI C, this is declared `int foo(void);`. With the flag

'**-fno-strict-prototype**', declaring functions with no arguments is equivalent to declaring its argument list to be untyped, i.e., `int foo();` is equivalent to saying `int foo(...);`.

-fnonnull-objects

Normally, GNU C++ makes conservative assumptions about objects reached through references. For example, the compiler must check that 'a' is not null in code like the following:

```
obj &a = g ();
a.f (2);
```

Checking that references of this sort have non-null values requires extra code, however, and it is unnecessary for many programs. You can use '**-fnonnull-objects**' to omit the checks for null, if your program doesn't require the default checking.

-fhandle-signatures
-fno-handle-signatures

These options control the recognition of the `signature` and `sigof` constructs for specifying abstract types. By default, these constructs are not recognized.

-fthis-is-variable

The incorporation of user-defined free store management into C++ has made assignment to `this` an anachronism. Therefore, by default GNU C++ treats the type of `this` in a member function of `class X` to be `X *const`. In other words, it is illegal to assign to `this` within a class member function. However, for backwards compatibility, you can invoke the old behavior by using '**-fthis-is-variable**'.

-g

Produce debugging information in the operating system's native format (for DBX or SDB or DWARF). GDB also can work with this debugging information. On most systems that use DBX format, '**-g**' enables use of extra debugging information that only GDB can use.

Unlike most other C compilers, GNU CC allows you to use '**-g**' with '**-O**'. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

-I*dir* Append directory *dir* to the list of directories searched for include files.

-L*dir* Add directory *dir* to the list of directories to be searched for '**-l**'.

-l*library* Use the library named *library* when linking. (C++ programs often require '**-lg++**' for successful linking.)

-nostdinc Do not search the standard system directories for header files. Only the directories you have specified with **-I** options (and the current directory, if appropriate) are searched.

-nostdinc++ Do not search for header files in the standard directories specific to C++, but do still search the other standard directories. (This option is used when building libg++.)

-O Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

-o *file* Place output in file *file*.

-S Stop after the stage of compilation proper; do not assemble. The output is an assembler code file for each non-assembler input file specified.

-traditional Attempt to support some aspects of traditional C compilers.

Specifically, for both C and C++ programs:

1. In the preprocessor, comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.

2. In the preprocessor, macro arguments are recognized within string constants in a macro definition (and their values are stringified, though without additional quote marks, when they appear in such a context). The preprocessor always considers a string constant to end at a newline.

3. The preprocessor does not predefine the macro `__STDC__` when you use '**-traditional**', but still predefines `__GNUC__` (since the GNU extensions indicated by `__GNUC__` are not affected by '**-traditional**'). If you need to write header files that work differently depending on whether '**-traditional**' is in use, by testing both of these predefined macros you can distinguish four situations: GNU C, traditional GNU C, other ANSI C compilers, and other old C compilers.

4. In the preprocessor, comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.

5. In the preprocessor, macro arguments are recognized within string constants in a macro definition (and their values are stringified, though without additional quote marks, when they appear in such a context). The preprocessor always considers a string constant to end at a newline.

6. The preprocessor does not predefine the macro `__STDC__` when you use '**-traditional**', but still predefines `__GNUC__` (since the GNU extensions indicated by `__GNUC__` are not affected by '**-traditional**'). If you need to write header files that work differently depending on whether '**-traditional**' is in use, by testing both of these predefined macros you can distinguish four situations: GNU C, traditional GNU C, other ANSI C compilers, and other old C compilers.

7. String "constants" are not necessarily constant; they are stored in writable space, and identical looking constants are allocated separately.

For C++ programs only (not C), '**-traditional**' has one additional effect: assignment to `this` is permitted. This is the same as the effect of '**-fthis-is-variable**'.

-U*macro* Undefine macro *macro*.

-Wall Issue warnings for conditions which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

-Wenum-clash Warn when converting between different enumeration types.

-Woverloaded-virtual
In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. Use this option to request warnings when a derived class declares a function that may be an erroneous attempt to define a virtual function: that is, warn when a function with the same name as a virtual function in the base class, but with a type signature that doesn't match any virtual functions from the base class.

-Wtemplate-debugging
When using templates in a C++ program, warn if debugging is not yet fully available.

-w Inhibit all warning messages.

+eN Control how virtual function definitions are used, in a fashion compatible with `cfront` 1.x.

PRAGMAS

Two '`#pragma`' directives are supported for GNU C++, to permit using the same header file for two purposes: as a definition of interfaces to a given object class, and as the full definition of the contents of that object class.

`#pragma interface`

Use this directive in header files that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing '`#pragma interface`' is included in a compilation, this auxiliary information will not be generated (unless the main input source file itself uses '`#pragma implementation`'). Instead, the object files will contain references to be resolved at link time.

`#pragma implementation`
`#pragma implementation "`*objects*`.h"`

Use this pragma in a main input file, when you want full output from included header files to be generated (and made globally visible). The included header file, in turn, should use '`#pragma interface`'. Backup copies of inline member functions, debugging information, and the internal tables used to implement virtual functions are all generated in implementation files.

If you use '`#pragma implementation`' with no argument, it applies to an include file with the same basename as your source file; for example, in '`allclass.cc`', '`#pragma implementation`' by itself is equivalent to '`#pragma implementation "allclass.h"`'. Use the string argument if you want a single implementation file to include code from multiple header files.

There is no way to split up the contents of a single header file into multiple implementation files.

FILES

`file.h`	C header (preprocessor) file
`file.i`	preprocessed C source file

`file.C`	C++ source file
`file.cc`	C++ source file
`file.cxx`	C++ source file
`file.s`	assembly language file
`file.o`	object file
`a.out`	link edited output
TMPDIR/`cc*`	temporary files
LIBDIR/`cpp`	preprocessor
LIBDIR/`cc1plus`	compiler
LIBDIR/`collect`	linker front end needed on some machines
LIBDIR/`libgcc.a`	GCC subroutine library
`/lib/crt[01n].o`	start-up routine
LIBDIR/`ccrt0`	additional start-up routine for C++
`/lib/libc.a`	standard C library, see `intro`(3)
`/usr/include`	standard directory for `#include` files
LIBDIR/`include`	standard gcc directory for `#include` files
LIBDIR/`g++-include`	additional g++ directory for `#include`

LIBDIR is usually `/usr/local/lib/`*machine*/*version*.

TMPDIR comes from the environment variable `TMPDIR` (default `/usr/tmp` if available, else `/tmp`).

SEE ALSO

Utility Programs -	`gcc, cpp, as, ld, gdb, adb, dbx, sdb`
Documents -	*Using and Porting GNU CC (for version 2.0)*, Richard M. Stallman;
	The C Preprocessor, Richard M. Stallman;
	Debugging with GDB: the GNU Source-Level Debugger, Richard M. Stallman and Roland H. Pesch;
	Using as: the GNU Assembler, Dean Elsner, Jay Fenlason & friends;
	gld: the GNU linker, Steve Chamberlain and Roland Pesch.

BUGS

For instructions on how to report bugs, see the GCC manual.

COPYING

AUTHORS

See the GNU CC Manual for the contributors to GNU CC.

NAME

gawk – pattern scanning and processing language

SYNOPSIS

gawk [-W *gawk_options* **] [-F***fs* **] [-v** *var*=*val* **] -f** *program_file* **[--]** *file ...*

gawk [-W *gawk_options* **] [-F***fs* **] [-v** *var*=*val* **] [--]** *program_text file ...*

DESCRIPTION

gawk is the GNU Project's implementation of the AWK programming language. It conforms to the definition of the language in the *POSIX 1003.2 Command Language And Utilities Standard* (Draft 11). This version, in turn, is based on the description in *The AWK Programming Language*, by Aho, Kernighan, and Weinberger, with the additional features defined in the System V Release 4 version of UNIX awk. gawk also provides some GNU-specific extensions.

The command line consists of options to gawk itself, the AWK program text (if not supplied via the **-f** option), and values to be made available in the ARGC and ARGV pre-defined AWK variables.

OPTIONS

gawk accepts the following options, which should be available on any implementation of the AWK language.

-F*fs* Use *fs* for the input field separator (the value of the FS predefined variable).

-v *var*=*val* Assign the value *val*, to the variable *var*, before execution of the program begins. Such variable values are available to the BEGIN block of an AWK program.

-f *program_file* Read the AWK program source from the file *program_file*, instead of from the first command line argument. Multiple **-f** options may be used.

`--`	Signal the end of options. This is useful to allow further arguments to the AWK program itself to start with a `-`. This is mainly for consistency with the argument parsing convention used by most other POSIX programs.

Following the POSIX standard, `gawk`-specific options are supplied via arguments to the **-W** option. Multiple **-W** options may be supplied, or multiple arguments may be supplied together if they are separated by commas, or enclosed in quotes and separated by white space. Case is ignored in arguments to the **-W** option.

The **-W** option accepts the following arguments:

`compat`	Run in compatibility mode. In compatibility mode, `gawk` behaves identically to UNIX `awk`; none of the GNU-specific extensions are recognized. See GNU Extensions, below, for more information.
`copyleft` `copyright`	Print the short version of the GNU copyright information message on the error output.
`lint`	Provide warnings about constructs that are dubious or non-portable to other AWK implementations.
`posix`	This turns on compatibility mode, with the following additional restrictions: • `\x` escape sequences are not recognized. • The synonym `func` for the keyword `function` is not recognized. • The operators `**` and `**=` cannot be used in place of `^` and `^=`.
`version`	Print version information for this particular copy of `gawk` on the error output. This is useful mainly for knowing if the current copy of `gawk` on your system is up to date with respect to whatever the Free Software Foundation is distributing.

Any other options are flagged as illegal, but are otherwise ignored.

AWK PROGRAM EXECUTION

An AWK program consists of a sequence of pattern-action statements and optional function definitions.

> *pattern* { *action statements* }
> `function` *name*(*parameter list*) { *statements* }

`gawk` first reads the program source from the *program_file*(s) if specified, or from the first non-option argument on the command line. The `-f` option may be used multiple times on the command line. `gawk` will read the program text as if all the *program_file*s had been concatenated together. This is useful for building libraries of AWK functions, without having to include them in each new AWK program that uses them. To use a library function in a file from a program typed in on the command line, specify `/dev/tty` as one of the *program_file*s, type your program, and end it with a `^D` (Control-D).

The environment variable `AWKPATH` specifies a search path to use when finding source files named with the `-f` option. If this variable does not exist, the default path is `.:/usr/lib/awk:/usr/local/lib/awk`. If a file name given to the `-f` option contains a `/` character, no path search is performed.

`gawk` executes AWK programs in the following order. First, `gawk` compiles the program into an internal form. Next, all variable assignments specified via the `-v` option are performed. Then, `gawk` executes the code in the `BEGIN` block(s) (if any), and then proceeds to read each file named in the `ARGV` array. If there are no files named on the command line, `gawk` reads the standard input.

If a filename on the command line has the form *var=val* it is treated as a variable assignment. The variable *var* will be assigned the value *val*. (This happens after any `BEGIN` block(s) have been run.) Command line variable assignment is most useful for dynamically assigning values to the variables AWK uses to control how input is broken into fields and records. It is also useful for controlling state if multiple passes are needed over a single data file.

If the value of a particular element of `ARGV` is empty (`""`), `gawk` skips over it.

For each line in the input, `gawk` tests to see if it matches any *pattern* in the AWK program. For each pattern that the line matches, the associated *action* is executed. The patterns are tested in the order they occur in the program.

Finally, after all the input is exhausted, `gawk` executes the code in the `END` block(s) (if any).

VARIABLES AND FIELDS

AWK variables are dynamic; they come into existence when they are first used. Their values are either floating-point numbers or strings, or both, depending upon how they are used. AWK also has one dimension arrays; multiply dimensioned arrays may be simulated. Several pre-defined variables are set as a program runs; these will be described as needed and summarized below.

Fields

As each input line is read, `gawk` splits the line into *fields*, using the value of the `FS` variable as the field separator. If `FS` is a single character, fields are separated by that character. Otherwise, `FS` is expected to be a full regular expression. In the special case that `FS` is a single blank, fields are separated by runs of blanks and/or tabs. Note that the value of `IGNORECASE` (see below) will also affect how fields are split when `FS` is a regular expression.

If the `FIELDWIDTHS` variable is set to a space separated list of numbers, each field is expected to have fixed width, and `gawk` will split up the record using the specified widths. The value of `FS` is ignored. Assigning a new value to `FS` overrides the use of `FIELDWIDTHS`, and restores the default behavior.

Each field in the input line may be referenced by its position, `$1`, `$2`, and so on. `$0` is the whole line. The value of a field may be assigned to as well. Fields need not be referenced by constants:

```
n = 5
print $n
```

prints the fifth field in the input line. The variable NF is set to the total number of fields in the input line.

References to non-existent fields (i.e. fields after $NF) produce the null-string. However, assigning to a non-existent field (e.g., $(NF+2)=5) will increase the value of NF, create any intervening fields with the null string as their value, and cause the value of $0 to be recomputed, with the fields being separated by the value of OFS.

Built-In Variables

AWK's built-in variables are:

ARGC	The number of command line arguments (does not include options to gawk, or the program source).
ARGV	Array of command line arguments. The array is indexed from 0 to ARGC – 1. Dynamically changing the contents of ARGV can control the files used for data.
CONVFMT	The conversion format for numbers, %.6g, by default.
ENVIRON	An array containing the values of the current environment. The array is indexed by the environment variables, each element being the value of that variable (e.g., ENVIRON[HOME] might be /u/arnold). Changing this array does not affect the environment seen by programs which gawk spawns via redirection or the system() function. (This may change in a future version of gawk.)
FIELDWIDTHS	A white-space separated list of fieldwidths. When set, gawk parses the input into fields of fixed width, instead of using the value of the FS variable as the field separator. The fixed field width facility is still experimental; expect the semantics to change as gawk evolves over time.
FILENAME	The name of the current input file. If no files are specified on the command line, the value of FILENAME is "–".
FNR	The input record number in the current input file.

`FS`	The input field separator, a blank by default.
`IGNORECASE`	Controls the case-sensitivity of all regular expression operations. If `IGNORECASE` has a non-zero value, then pattern matching in rules, field splitting with `FS`, regular expression matching with `~` and `!~`, and the `gsub()`, `index()`, `match()`, `split()`, and `sub()` predefined functions will all ignore case when doing regular expression operations. Thus, if `IGNORECASE` is not equal to zero, `/aB/` matches all of the strings `ab`, `aB`, `Ab`, and `AB`. As with all AWK variables, the initial value of `IGNORECASE` is zero, so all regular expression operations are normally case-sensitive.
`NF`	The number of fields in the current input record.
`NR`	The total number of input records seen so far.
`OFMT`	The output format for numbers, `%.6g`, by default.
`OFS`	The output field separator, a blank by default.
`ORS`	The output record separator, by default a newline.
`RS`	The input record separator, by default a newline. `RS` is exceptional in that only the first character of its string value is used for separating records. (This will probably change in a future release of `gawk`.) If `RS` is set to the null string, then records are separated by blank lines. When `RS` is set to the null string, then the newline character always acts as a field separator, in addition to whatever value `FS` may have.
`RSTART`	The index of the first character matched by `match()`; 0 if no match.
`RLENGTH`	The length of the string matched by `match()`; –1 if no match.
`SUBSEP`	The character used to separate multiple subscripts in array elements, by default `\034`.

Arrays

Arrays are subscripted with an expression between square brackets (`[`and`]`). If the expression is an expression list (*expr*, *expr*...) then the array subscript is a string consisting of the concatenation of the (string) value of each expression, separated by the value of the `SUBSEP` variable. This facility is used to simulate multiply dimensioned arrays. For example:

```
i = "A" ; j = "B" ; k = "C"
x[i, j, k] = "hello, world\n"
```

assigns the string `hello, world\n` to the element of the array `x` which is indexed by the string `A\034B\034C`. All arrays in AWK are associative, i.e. indexed by string values.

The special operator `in` may be used in an `if` or `while` statement to see if an array has an index consisting of a particular value.

```
if (val in array)
print array[val]
```

If the array has multiple subscripts, use `(i,j)inarray`.

The `in` construct may also be used in a `for` loop to iterate over all the elements of an array.

An element may be deleted from an array using the `delete` statement.

Variable Typing And Conversion

Variables and fields may be (floating point) numbers, or strings, or both. How the value of a variable is interpreted depends upon its context. If used in a numeric expression, it will be treated as a number, if used as a string it will be treated as a string.

To force a variable to be treated as a number, add 0 to it; to force it to be treated as a string, concatenate it with the null string.

When a string must be converted to a number, the conversion is accomplished using `atof()`. A number is converted to a string by using the value of `CONVFMT` as a format string for `sprintf()`, with the numeric value of the variable as the argument. However, even though all numbers in AWK are floating-point, integral values are *always* converted as integers. Thus, given

```
CONVFMT = "%2.2f"
a = 12
b = a ""
```

the variable `b` has a value of `12` and not `12.00`.

`gawk` performs comparisons as follows: If two variables are numeric, they are compared numerically. If one value is numeric and the other has a string value that is a "numeric string," then comparisons are also done numerically. Otherwise, the numeric value is converted to a string and a string comparison is performed. Two strings are compared, of course, as strings. According to the POSIX standard (Draft 11), even if two strings are numeric strings, a numeric comparison is performed. However, this is clearly incorrect, and `gawk` does not do this.

Uninitialized variables have the numeric value `0` and the string value "" (the null, or empty, string).

PATTERNS AND ACTIONS

AWK is a line oriented language. The pattern comes first, and then the action. Action statements are enclosed in `{` and `}`. Either the pattern may be missing, or the action may be missing, but, of course, not both. If the pattern is missing, the action will be executed for every single line of input. A missing action is equivalent to

```
{ print }
```

which prints the entire line.

Comments begin with the `#` character, and continue until the end of the line. Blank lines may be used to separate statements. Normally, a statement ends with a newline, however, this is not the case for lines ending in a `,`, `{`, `?`, `:`, `&&`, or `||`. Lines ending in `do` or `else` also have their statements automatically continued on the following line. In other cases, a line can be continued by ending it with a `\`, in which case the newline will be ignored.

Multiple statements may be put on one line by separating them with a `;`. This applies to both the statements within the action part of a pattern-action pair (the usual case), and to the pattern-action statements themselves.

Patterns

AWK patterns may be one of the following:

```
BEGIN
END
/regular expression/
relational expression
pattern && pattern
pattern || pattern
pattern ? pattern : pattern
(pattern)
! pattern
pattern1, pattern2
```

`BEGIN` and `END` are two special kinds of patterns which are not tested against the input. The action parts of all `BEGIN` patterns are merged as if all the statements had been written in a single `BEGIN` block. They are executed before any of the input is read. Similarly, all the `END` blocks are merged, and executed when all the input is exhausted (or when an `exit` statement is executed). `BEGIN` and `END` patterns cannot be combined with other patterns in pattern expressions. `BEGIN` and `END` patterns cannot have missing action parts.

For /*regular expression*/ patterns, the associated statement is executed for each input line that matches the regular expression. Regular expressions are the same as those in `egrep`, and are summarized below.

A *relational expression* may use any of the operators defined below in the section on actions. These generally test whether certain fields match certain regular expressions.

The `&&`, `||`, and `!` operators are logical `AND`, logical `OR`, and logical `NOT`, respectively, as in C. They do short-circuit evaluation, also as in C, and are used for combining more primitive pattern expressions. As in most languages, parentheses may be used to change the order of evaluation.

The `?:` operator is like the same operator in C. If the first pattern is true then the pattern used for testing is the second pattern, otherwise it is the third. Only one of the second and third patterns is evaluated.

The *pattern1*`,` *pattern2* form of an expression is called a range pattern. It matches all input records starting with a line that matches *pattern1*, and continuing until a record that matches *pattern2*, inclusive. It does not combine with any other sort of pattern expression.

Regular Expressions

Regular expressions are the extended kind found in `egrep`. They are composed of characters as follows:

c matches the non-metacharacter *c*.

`\`*c*	Matches the literal character *c*.
`.`	Matches any character except newline.
`^`	Matches the beginning of a line or a string.
`$`	Matches the end of a line or a string.
`[`*abc...*`]`	Character class, matches any of the characters *abc....*
`[^`*abc...*`]`	Negated character class, matches any character except *abc...* and newline.
r1|*r2*	Alternation: matches either *r1* or *r2*.
r1r2	Concatenation: matches *r1*, and then *r2*.
r+	Matches one or more *r*'s.
*r**	Matches zero or more *r*'s.
r?	Matches zero or one *r*'s.
`(`*r*`)`	Grouping: matches *r*.

The escape sequences that are valid in string constants (see below) are also legal in regular expressions.

Actions

Action statements are enclosed in braces, `{` and `}`. Action statements consist of the usual assignment, conditional, and looping statements found in most languages. The operators, control statements, and input/output statements available are patterned after those in C.

Operators

The operators in AWK, in order of increasing precedence, are

`= += -=`	
`*= /= %= ^=`	Assignment. Both absolute assignment (*var=value*) and operator-assignment (the other forms) are supported.
`?:`	The C conditional expression. This has the form *expr1* `?` *expr2* `:` *expr3*.If *expr1* is true, the value of the expression is *expr2*, otherwise it is *expr3*. Only one of *expr2* and *expr3* is evaluated.
`||`	Logical `OR`.
`&&`	Logical `AND`.
`~ !~`	Regular expression match, negated match. `NOTE:` Do not use a constant regular expression (`/foo/`) on the left-hand side of a `~` or `!~`. Only use one on the right-hand side. The expression `/foo/~`*exp* has the same meaning as `(($0 ~ /foo/) ~` *exp*`)`. This is usually not what was intended.
`< >`	
`<= >=`	
`!= ==`	The regular relational operators.
blank	String concatenation.
`+ -`	Addition and subtraction.
`* / %`	Multiplication, division, and modulus.

`+ - !`	Unary plus, unary minus, and logical negation.
`^`	Exponentiation (`**` may also be used, and `**=` for the assignment operator).
`++ --`	Increment and decrement, both prefix and postfix.
`$`	Field reference.

Control Statements

The control statements are as follows:

```
if (condition) statement [ else statement ]
while (condition) statement
do statement while (condition)
for (expr1; expr2; expr3) statement
for (var in array) statement
break
continue
delete array[index]
exit [ expression ]
{ statements }
```

I/O Statements

The input/output statements are as follows:

`close(`*filename*`)`	Close file (or pipe, see below).
`getline`	Set `$0` from next input record; set `NF`, `NR`, `FNR`.
`getline <`*file*	Set `$0` from next record of *file*; set `NF`.
`getline` *var*	Set *var* from next input record; set `NF`, `FNR`.
`getline` *var* `<`*file*	Set *var* from next record of *file*.
`next`	Stop processing the current input record. The next input record is read and processing starts over with the first pattern in the AWK program. If the end of the input data is reached, the `END` block(s), if any, are executed.

`next file` Stop processing the current input file. The next input record read comes from the next input file. `FILENAME` is updated, `FNR` is reset to 1, and processing starts over with the first pattern in the AWK program. If the end of the input data is reached, the `END` block(s), if any, are executed.

`print` Prints the current record.

`print` *expr-list* Prints expressions.

`print` *expr-list* >*file*
Prints expressions on *file*.

`printf` *fmt, expr-list*
Format and print.

`printf` *fmt, expr-list* >*file*
Format and print on *file*.

`system(`*cmd-line*`)` Execute the command *cmd-line*, and return the exit status. (This may not be available on non-POSIX systems.)

Other input/output redirections are also allowed. For `print` and `printf`, >>*file* appends output to the *file*, while | *command* writes on a pipe. In a similar fashion, *command*|`getline` pipes into `getline`. `Getline` will return `0` on end of file, and `-1` on an error.

The `printf` Statement

The AWK versions of the `printf` statement and `sprintf()` function (see below) accept the following conversion specification formats:

`%c` An ASCII character. If the argument used for `%c` is numeric, it is treated as a character and printed. Otherwise, the argument is assumed to be a string, and the only first character of that string is printed.

`%d` A decimal number (the integer part).

`%i` Just like `%d`.

`%e`	A floating point number of the form `[-]d.ddddddE[+-]dd`.
`%f`	A floating point number of the form `[-]ddd.dddddd`.
`%g`	Use `e` or `f` conversion, whichever is shorter, with nonsignificant zeros suppressed.
`%o`	An unsigned octal number (again, an integer).
`%s`	A character string.
`%x`	An unsigned hexadecimal number (an integer).
`%X`	Like `%x`, but using `ABCDEF` instead of `abcdef`.
`%%`	A single `%` character; no argument is converted.

Additionally, the following optional parameters that may lie between the `%` and the control letter:

–	The expression should be left-justified within its field.
width	The field should be padded to this width. If the number has a leading zero, then the field will be padded with zeros. Otherwise it is padded with blanks.
.prec	A *number* `indicating` *the* `maximum` *width* `of` *strings* `or` *digits* `to` *the* `right` of the decimal point.

The dynamic *width* and *.prec* capabilities of the ANSI C `printf()` routines are supported. A `*` in place of either the *width* or *.prec* specifications will cause their values to be taken from the argument list to `printf` or `sprintf()`.

Special File Names

When doing I/O redirection from either `print` or `printf` into a file, or via `getline` from a file, `gawk` recognizes certain special filenames internally. These filenames allow access to open file descriptors inherited from `gawk`'s parent process (usually the shell). The filenames are:

`/dev/stdin`	The standard input.
`/dev/stdout`	The standard output.

`/dev/stderr`	The standard error output.
`/dev/fd/`*n*	The file associated with the open file descriptor *n*.

These are particularly useful for error messages. For example:

```
print "You blew it!" > "/dev/stderr"
```

whereas you would otherwise have to use

```
print "You blew it!" | "cat 1>&2"
```

These file names may also be used on the command line to name data files.

Numeric Functions

AWK has the following pre-defined arithmetic functions:

`atan2(`*y*`,`*x*`)`	Returns the arctangent of *y*/*x* in radians.
`cos(`*expr*`)`	Returns the cosine in radians.
`exp(`*expr*`)`	The exponential function.
`int(`*expr*`)`	Truncates to integer.
`log(`*expr*`)`	The natural logarithm function.
`rand()`	Returns a random number between 0 and 1.
`sin(`*expr*`)`	Returns the sine in radians.
`sqrt(`*expr*`)`	The square root function.
`srand(`*expr*`)`	Use *expr* as a new seed for the random number generator. If no *expr* is provided, the time of day will be used. The return value is the previous seed for the random number generator.

String Functions

AWK has the following pre-defined string functions:

`gsub(`*r*`, `*s*`, `*t*`)`	For each substring matching the regular expression *r* in the string *t*, substitute the string *s*, and return the number of substitutions. If *t* is not supplied, use `$0`.

`index(`*s*`,`*t*`)` Returns the index of the string *t* in the string *s*, or 0 if *t* is not present.

`length(`*s*`)` Returns the length of the string *s*, or the length of `$0` if *s* is not supplied.

`match(`*s*`,`*r*`)` Returns the position in *s* where the regular expression *r* occurs, or 0 if *r* is not present, and sets the values of `RSTART` and `RLENGTH`.

`split(`*s*`, `*a*`, `*r*`)`
Splits the string *s* into the array *a* on the regular expression *r*, and returns the number of fields. If *r* is omitted, `FS` is used instead.

`sprintf(`*fmt*`, `*expr-list*`)`
Prints *expr-list* according to *fmt*, and returns the resulting string.

`sub(`*r*`, `*s*`, `*t*`)` Just like `gsub()`, but only the first matching substring is replaced.

`substr(`*s*`, `*i*`, `*n*`)`
Returns the *n*-character substring of *s* starting at *i*. If *n* is omitted, the rest of *s* is used.

`tolower(`*str*`)` Returns a copy of the string *str*, with all the upper-case characters in *str* translated to their corresponding lower-case counterparts. Non-alphabetic characters are left unchanged.

`toupper(`*str*`)` Returns a copy of the string *str*, with all the lower-case characters in *str* translated to their corresponding upper-case counterparts. Non-alphabetic characters are left unchanged.

Time Functions

Since one of the primary uses of AWK programs is processing log files that contain time stamp information, `gawk` provides the following two functions for obtaining time stamps and formatting them.

`systime()` Returns the current time of day as the number of seconds since the Epoch (Midnight UTC, January 1, 1970 on POSIX systems).

`strftime(`*format*, *timestamp*`)`
Formats *timestamp* according to the specification in *format.* The *timestamp* should be of the same form as returned by `systime()`. If *timestamp* is missing, the current time of day is used. See the specification for the `strftime()` function in ANSI C for the format conversions that are guaranteed to be available. A public-domain version of `strftime()` and a man page for it are shipped with `gawk`; if that version was used to build `gawk`, then all of the conversions described in that man page are available to `gawk`.

String Constants

String constants in AWK are sequences of characters enclosed between double quotes ("). Within strings, certain escape sequences are recognized, as in C. These are:

`\a` The "alert" character; usually the ASCII BEL character.

`\b` Backspace.

`\f` Form-feed.

`\n` New line.

`\r` Carriage return.

`\t` Horizontal tab.

`\v` Vertical tab.

`\x` *hex digits* The character represented by the string of hexadecimal digits following the `\x`. As in ANSI C, all following hexadecimal digits are considered part of the escape sequence. (This feature should tell us something about language design by committee.) E.g., \x1B is the ASCII ESC (escape) character.

`\`*ddd*	The character represented by the 1-, 2-, or 3-digit sequence of octal digits. E.g. `\033` is the ASCII ESC (escape) character.
`\`*c*	The literal character *c*.

The escape sequences may also be used inside constant regular expressions (e.g., `/[\t\f\n\r\v]/` matches whitespace characters).

FUNCTIONS

Functions in AWK are defined as follows:

`function` *name*`(` *parameter list* `) {` *statements* `}`

Functions are executed when called from within the action parts of regular pattern-action statements. Actual parameters supplied in the function call are used to instantiate the formal parameters declared in the function. Arrays are passed by reference, other variables are passed by value.

Since functions were not originally part of the AWK language, the provision for local variables is rather clumsy: They are declared as extra parameters in the parameter list. The convention is to separate local variables from real parameters by extra spaces in the parameter list. For example:

```
function f(p, q, a, b) { # a & b are local
         ..... }
/abc/ { ... ; f(1, 2) ; ... }
```

The left parenthesis in a function call is required to immediately follow the function name, without any intervening white space. This is to avoid a syntactic ambiguity with the concatenation operator. This restriction does not apply to the built-in functions listed above.

Functions may call each other and may be recursive. Function parameters used as local variables are initialized to the null string and the number zero upon function invocation.

The word `func` may be used in place of `function`.

EXAMPLES

Print and sort the login names of all users:

```
BEGIN    { FS = ":" }
{ print $1 | "sort" }
```

Count lines in a file:

```
         { nlines++ }
END      { print nlines }
```

Precede each line by its number in the file:

```
{ print FNR, $0 }
```

Concatenate and line number (a variation on a theme):

```
{ print NR, $0 }
```

SEE ALSO

Utility Programs - `egrep`

Documents - *The AWK Programming Language*, Alfred V. Aho, Brian W. Kernighan, Peter J. Weinberger, Addison-Wesley, 1988. ISBN 0-201-07981-X.

The GAWK Manual, Edition 0.14, published by the Free Software Foundation, 1992.

POSIX COMPATIBILITY

A primary goal for `gawk` is compatibility with the POSIX standard, as well as with the latest version of UNIX `awk`. To this end, `gawk` incorporates the following user visible features which are not described in the AWK book, but are part of `awk` in System V Release 4, and are in the POSIX standard.

The **–v** option for assigning variables before program execution starts is new. The book indicates that command line variable assignment happens when `awk` would otherwise open the argument as a file, which is after the `BEGIN` block is executed. However, in earlier implementations, when such an assignment appeared before any file names, the assignment would happen *before* the `BEGIN` block was run. Applications came to depend on this "feature." When `awk` was changed to match its documentation, this option was added to accommodate applications that depended upon the old behavior. (This feature was agreed upon by both the AT&T and GNU developers.)

The **-W** option for implementation specific features is from the POSIX standard.

When processing arguments, `gawk` uses the special option—to signal the end of arguments, and warns about, but otherwise ignores, undefined options.

The AWK book does not define the return value of `srand()`. The System V Release 4 version of UNIX `awk` (and the POSIX standard) has it return the seed it was using, to allow keeping track of random number sequences. Therefore `srand()` in `gawk` also returns its current seed.

Other new features are: The use of multiple **-f** options (from MKS `awk`); the `ENVIRON` array; the `\a`, and `\v` escape sequences (done originally in `gawk` and fed back into AT&T's); the `tolower()` and `toupper()` built-in functions (from AT&T); and the ANSI C conversion specifications in `printf` (done first in AT&T's version).

GNU EXTENSIONS

`gawk` has some extensions to POSIX `awk`. They are described in this section. All the extensions described here can be disabled by invoking `gawk` with the **-W** `compat` option.

The following features of `gawk` are not available in POSIX `awk`.

- The `systime()` and `strftime()` functions.
- The special file names available for I/O redirection are not recognized.
- The `IGNORECASE` variable and its side-effects are not available.
- The `FIELDWIDTHS` variable and fixed width field splitting.
- No path search is performed for files named via the **-f** option. Therefore the `AWKPATH` environment variable is not special.

- The use of `next file` to abandon processing of the current input file.

The AWK book does not define the return value of the `close()` function. `gawk`'s `close()` returns the value from `fclose()`, or `pclose()`, when closing a file or pipe, respectively.

When `gawk` is invoked with the **-W** `compat` option, if the *fs* argument to the **-F** option is `t`, then `FS` will be set to the tab character. Since this is a rather ugly special case, it is not the default behavior. This behavior also does not occur if `-Wposix` has been specified.

HISTORICAL FEATURES

There are two features of historical AWK implementations that `gawk` supports. First, it is possible to call the `length()` built-in function not only with no argument, but even without parentheses! Thus,

```
a = length
```

is the same as either of

```
a = length()
a = length($0)
```

This feature is marked as "deprecated" in the POSIX standard, and `gawk` will issue a warning about its use if `-Wlint` is specified on the command line.

The other feature is the use of the `continue` statement outside the body of a `while`, `for`, or `do` loop. Traditional AWK implementations have treated such usage as equivalent to the `next` statement. `gawk` will support this usage if `-Wposix` has not been specified.

BUGS

The **-F** option is not necessary given the command line variable assignment feature; it remains only for backwards compatibility.

VERSION INFORMATION

This man page documents `gawk`, version 2.14.

For the 2.14 version of `gawk`, the **−c**, **−V**, **−C**, **−D**, **−a**, and **−e** options of the 2.11 version are recognized. However, `gawk` will print a warning message, and these options will go away in the 2.15 version.

AUTHORS

The original version of UNIX `awk` was designed and implemented by Alfred Aho, Peter Weinberger, and Brian Kernighan of AT&T Bell Labs. Brian Kernighan continues to maintain and enhance it.

Paul Rubin and Jay Fenlason, of the Free Software Foundation, wrote `gawk`, to be compatible with the original version of `awk` distributed in Seventh Edition UNIX. John Woods contributed a number of bug fixes. David Trueman, with contributions from Arnold Robbins, made `gawk` compatible with the new version of UNIX `awk`.

The initial DOS port was done by Conrad Kwok and Scott Garfinkle. Scott Deifik is the current DOS maintainer. Pat Rankin did the port to VMS, and Michal Jaegermann did the port to the Atari ST.

ACKNOWLEDGEMENTS

Brian Kernighan of Bell Labs provided valuable assistance during testing and debugging. We thank him.

COPYRIGHT

NAME

`gcc, cc, g++, c++` – GNU C and C++ Compiler (v2.6)

SYNOPSIS

gcc [*option* | *filename*]...

g++ [*option* | *filename*]...

DESCRIPTION

The information in this manual page is an extract from the full documentation of the GNU C compiler, and is limited to the meaning of the options. For complete and current documentation, refer to the manual *Using and Porting GNU CC (for version 2.0)*.

The C and C++ compilers are integrated. Both process input files through one or more of four stages: preprocessing, compilation, assembly, and linking. Source file name suffixes identify the source language, but which name you use for the compiler governs default assumptions:

`gcc`	assumes preprocessed (`.i`) files are C and assumes C style linking.
`g++`	assumes preprocessed (`.i`) files are C++ and assumes C++ style linking.

Suffixes of source file names indicate the language and kind of processing to be done:

`.c`	C source; preprocess, compile, assemble
`.C`	C++ source; preprocess, compile, assemble
`.cc`	C++ source; preprocess, compile, assemble
`.cxx`	C++ source; preprocess, compile, assemble
`.m`	Objective-C source; preprocess, compile, assemble
`.i`	preprocessed C; compile, assemble
`.ii`	preprocessed C++; compile, assemble

`.s`	Assembler source; assemble
`.S`	Assembler source; preprocess, assemble
`.h`	Preprocessor file; not usually named on command line

Files with other suffixes are passed to the linker. Common cases include:

`.o`	Object file
`.a`	Archive file

Linking is always the last stage unless you use one of the **–c**, **–S**, or **–E** options to avoid it (or unless compilation errors stop the whole process). For the link stage, all `.o` files corresponding to source files, **–l** libraries, unrecognized file names (including named `.o` object files and `.a` archives) are passed to the linker in command-line order.

OPTIONS

Options must be separate: '**–dr**' is quite different from '**–d –r** '.

Most '**–f**' and '**–W**' options have two contrary forms: **–f***name* and **–fno–***name* (or **–W***name* and **–Wno–***name*). Only the non-default forms are shown here.

Here is a summary of all the options, grouped by type. Explanations are in the following sections.

Overall Options
–c –S –E –o *file* **–pipe –v –x** *language*

Lynx POSIX Options
–mthreads –mposix4d9

Language Options
–ansi –fall–virtual –fcond–mismatch
–fdollars–in–identifiers –fenum–int–equiv
–fexternal–templates –fno–asm –fno–builtin
–fno–strict–prototype –fsigned–bitfields
–fsigned–char –fthis–is–variable

-funsigned-bitfields -funsigned-char
-fwritable-strings -traditional
-traditional-cpp -trigraphs

Warning Options

-fsyntax-only -pedantic -pedantic-errors -w -W
-Wall -Waggregate-return -Wcast-align
-Wcast-qual -Wchar-subscript -Wcomment
-Wconversion -Wenum-clash -Werror -Wformat
-Wid-clash-*len* **-Wimplicit -Winline**
-Wmissing-prototypes -Wmissing-declarations
-Wnested-externs -Wno-import -Wparentheses
-Wpointer-arith -Wredundant-decls
-Wreturn-type -Wshadow -Wstrict-prototypes
-Wswitch -Wtemplate-debugging -Wtraditional
-Wtrigraphs -Wuninitialized -Wunused
-Wwrite-strings

Debugging Options

-a -d*letters* **-fpretend-float -g -g***level* **-gcoff**
-gxcoff -gxcoff+ -gdwarf -gdwarf+ -gstabs
-gstabs+ -ggdb -p -pg -save-temps
-print-file-name=*library*
-print-libgcc-file-name
-print-prog-name=*program*

Optimization Options

-fcaller-saves -fcse-follow-jumps
-fcse-skip-blocks -fdelayed-branch
-felide-constructors
-fexpensive-optimizations -ffast-math
-ffloat-store -fforce-addr -fforce-mem
-finline-functions -fkeep-inline-functions
-fmemoize-lookups -fno-default-inline
-fno-defer-pop -fno-function-cse -fno-inline
-fno-peephole -fomit-frame-pointer
-frerun-cse-after-loop -fschedule-insns
-fschedule-insns2 -fstrength-reduce
-fthread-jumps -funroll-all-loops
-funroll-loops -O -O2

Preprocessor Options
: **-A***assertion* **-C** **-dD** **-dM** **-dN** **-D***macro*[=*defn*] **-E** **-H** **-idirafter** *dir* **-include** *file* **-imacros** *file* **-iprefix** *file* **-iwithprefix** *dir* **-M** **-MD** **-MM** **-MMD** **-nostdinc** **-P** **-U***macro* **-undef**

Assembler Option
: **-Wa,***option*

Linker Options
: **-l***library* **-nostartfiles** **-nostdlib** **-static** **-shared** **-symbolic** **-Xlinker** *option* **-Wl,***option* **-u** *symbol*

Directory Options
: **-B***prefix* **-I***dir* **-I-** **-L***dir*

Target Options
: **-b** *machine* **-V** *version*

Configuration Dependent Options

x86 Options
-m486 **-mno-486** **-msoft-float** **-mno-fp-ret-in-387**

68k Options
-m68000 **-m68020** **-m68020-40** **-m68030** **-m68040** **-m68881** **-mbitfield** **-mc68000** **-mc68020** **-mfpa** **-mnobitfield** **-mrtd** **-mshort** **-msoft-float**

PowerPC Options
-mfp-in-toc **-mno-fop-in-toc**

SPARC Options
-mepilogue **-mfpu** **-mhard-float** **-mno-fpu**

Code Generation Options
: **-fcall-saved-***reg* **-fcall-used-***reg* **-ffixed-***reg* **-finhibit-size-directive** **-fnonnull-objects** **-fno-common** **-fcommon** **-fno-ident** **-fno-gnu-linker** **-fpcc-struct-return** **-fpic** **-fPIC** **-freg-struct-return** **-fshared-data** **-fshort-enums** **-fshort-double** **-fvolatile** **-fvolatile-global** **-fverbose-asm**

Overall Options

–x *language*
: Specify explicitly the *language* for the following input files (rather than choosing a default based on the file name suffix) . This option applies to all following input files until the next '**–x**' option. Possible values of *language* are 'c', 'objective-c', 'c-header', 'c++', 'cpp-output', 'assembler', and 'assembler-with-cpp'.

–x none
: Turn off any specification of a language, so that subsequent files are handled according to their file name suffixes (as they are if '**–x**' has not been used at all).

If you want only some of the four stages (preprocess, compile, assemble, link), you can use '**–x**' (or filename suffixes) to tell gcc where to start, and one of the options '**–c**', '**–S**', or '**–E**' to say where gcc is to stop. Note that some combinations (for example, '-x cpp-output -E') instruct gcc to do nothing at all.

–c
: Compile or assemble the source files, but do not link. The compiler output is an object file corresponding to each source file.

By default, GCC makes the object file name for a source file by replacing the suffix '.c', '.i', '.s', etc., with '.o'. Use **–o** to select another name.

GCC ignores any unrecognized input files (those that do not require compilation or assembly) with the **–c** option.

–S
: Stop after the stage of compilation proper; do not assemble. The output is an assembler code file for each non-assembler input file specified.

By default, GCC makes the assembler file name for a source file by replacing the suffix '.c', '.i', etc., with '.s'. Use **–o** to select another name.

GCC ignores any input files that don't require compilation.

-E Stop after the preprocessing stage; do not run the compiler proper. The output is preprocessed source code, which is sent to the standard output.

GCC ignores input files which don't require preprocessing.

-o *file* Place output in file *file*. This applies regardless to whatever sort of output GCC is producing, whether it be an executable file, an object file, an assembler file or preprocessed C code.

Since only one output file can be specified, it does not make sense to use '**-o**' when compiling more than one input file, unless you are producing an executable file as output.

If you do not specify '**-o**', the default is to put an executable file in '`a.out`', the object file for '*source*.*suffix*' in '*source*.`o`', its assembler file in '*source*.`s`', and all preprocessed C source on standard output.

-v Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.

-pipe Use pipes rather than temporary files for communication between the various stages of compilation. This fails to work on some systems where the assembler cannot read from a pipe; but the GNU assembler has no trouble.

Lynx POSIX Options

Applications compiled with the GNU C compilers get POSIX 1003.1 (C language standard) and POSIX 1003.1b (Real-time extensions) support by default in LynxOS.

The following options add support for additional POSIX facilities:

-mthreads POSIX 1003.4a Draft 4 (Threads extensions).

–mposix4d9 POSIX 1003.4 Draft 9. This option is mutually exclusive of POSIX 1003.1b facilities and will be removed in a future release of LynxOS. Applications that continue to use POSIX 1003.4 Draft 9 facilities should be migrated to POSIX 1003.1b.

Language Options

The following options control the dialect of C that the compiler accepts:

–ansi Support all ANSI standard C programs.

This turns off certain features of GNU C that are incompatible with ANSI C, such as the `asm`, `inline` and `typeof` keywords, and predefined macros such as `unix` and `vax` that identify the type of system you are using. It also enables the undesirable and rarely used ANSI trigraph feature, and disallows '`$`' as part of identifiers.

The alternate keywords `__asm__`, `__extension__`, `__inline__` and `__typeof__` continue to work despite '**–ansi**'. You would not want to use them in an ANSI C program, of course, but it is useful to put them in header files that might be included in compilations done with '**–ansi**'. Alternate predefined macros such as `__unix__` and `__vax__` are also available, with or without '**–ansi**'.

The '**–ansi**' option does not cause non-ANSI programs to be rejected gratuitously. For that, '**–pedantic**' is required in addition to '**–ansi**'.

The preprocessor predefines a macro `__STRICT_ANSI__` when you use the '**–ansi**' option. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ANSI standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things.

-fno-asm Do not recognize asm, inline or typeof as a keyword. These words may then be used as identifiers. You can use __asm__, __inline__ and __typeof__ instead. '**-ansi**' implies '**-fno-asm**'.

-fno-builtin Don't recognize built-in functions that do not begin with two leading underscores. Currently, the functions affected include _exit, abort, abs, alloca, cos, exit, fabs, labs, memcmp, memcpy, sin, sqrt, strcmp, strcpy, and strlen.

The '**-ansi**' option prevents alloca and _exit from being builtin functions.

-fno-strict-prototype
Treat a function declaration with no arguments, such as 'int foo ();', as C would treat it—as saying nothing about the number of arguments or their types (C++ only). Normally, such a declaration in C++ means that the function foo takes no arguments.

-trigraphs Support ANSI C trigraphs. The '**-ansi**' option implies '**-trigraphs**'.

-traditional Attempt to support some aspects of traditional C compilers. For details, see the GNU C manual; the duplicate list here has been deleted so that we won't get complaints when it is out of date.

But one note about C++ programs only (not C). '**-traditional**' has one additional effect for C++: assignment to this is permitted. This is the same as the effect of '**-fthis-is-variable**'.

-traditional-cpp
Attempt to support some aspects of traditional C preprocessors. This includes the items that specifically mention the preprocessor above, but none of the other effects of '**-traditional**'.

-fdollars-in-identifiers
Permit the use of '`$`' in identifiers (C++ only). You can also use '**-fno-dollars-in-identifiers**' to explicitly prohibit use of '`$`'. (GNU C++ allows '`$`' by default on some target systems but not others.)

-fenum-int-equiv
Permit implicit conversion of `int` to enumeration types (C++ only). Normally GNU C++ allows conversion of `enum` to `int`, but not the other way around.

-fexternal-templates
Produce smaller code for template declarations, by generating only a single copy of each template function where it is defined (C++ only). To use this option successfully, you must also mark all files that use templates with either '`#pragma implementation`' (the definition) or '`#pragma interface`' (declarations).

When your code is compiled with '**-fexternal-templates**', all template instantiations are external. You must arrange for all necessary instantiations to appear in the implementation file; you can do this with a `typedef` that references each instantiation needed. Conversely, when you compile using the default option '**-fno-external-templates**', all template instantiations are explicitly internal.

-fall-virtual Treat all possible member functions as virtual, implicitly. All member functions (except for constructor functions and `new` or `delete` member operators) are treated as virtual functions of the class where they appear.

This does not mean that all calls to these member functions will be made through the internal table of virtual functions. Under some circumstances, the compiler can determine that a call to a given virtual function can be made directly; in these cases the calls are direct in any case.

-fcond-mismatch

Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void.

-fthis-is-variable

Permit assignment to `this` (C++ only). The incorporation of user-defined free store management into C++ has made assignment to '`this`' an anachronism. Therefore, by default it is invalid to assign to `this` within a class member function. However, for backwards compatibility, you can make it valid with '**-fthis-is-variable**'.

-funsigned-char

Let the type `char` be unsigned, like `unsigned char`.

Each kind of machine has a default for what `char` should be. It is either like `unsigned char` by default or like `signed char` by default.

Ideally, a portable program should always use `signed char` or `unsigned char` when it depends on the signedness of an object. But many programs have been written to use plain `char` and expect it to be signed, or expect it to be unsigned, depending on the machines they were written for. This option, and its inverse, let you make such a program work with the opposite default.

The type `char` is always a distinct type from each of `signed char` and `unsigned char`, even though its behavior is always just like one of those two.

-fsigned-char Let the type `char` be signed, like `signed char`.

Note that this is equivalent to '**-fno-unsigned-char**', which is the negative form of '**-funsigned-char**'. Likewise, '**-fno-signed-char**' is equivalent to '**-funsigned-char**'.

–fsigned–bitfields
–funsigned–bitfields
–fno–signed–bitfields
–fno–unsigned–bitfields

These options control whether a bitfield is signed or unsigned, when declared with no explicit '`signed`' or '`unsigned`' qualifier. By default, such a bitfield is signed, because this is consistent: the basic integer types such as `int` are signed types.

However, when you specify '**–traditional**', bitfields are all unsigned no matter what.

–fwritable–strings

Store string constants in the writable data segment and don't uniquize them. This is for compatibility with old programs which assume they can write into string constants. '**–traditional**' also has this effect.

Writing into string constants is a very bad idea; *constants* should be constant.

Preprocessor Options

These options control the C preprocessor, which is run on each C source file before actual compilation.

If you use the '**–E**' option, GCC does nothing except preprocessing. Some of these options make sense only together with '**–E**' because they cause the preprocessor output to be unsuitable for actual compilation.

–include *file*

Process *file* as input before processing the regular input file. In effect, the contents of *file* are compiled first. Any '**–D**' and '**–U**' options on the command line are always processed before '**–include** *file*', regardless of the order in which they are written. All the '**–include**' and '**–imacros**' options are processed in the order in which they are written.

–imacros *file*

Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of '**–imacros** *file*' is to make the macros defined in *file* available for use in the main input. The preprocessor evaluates any '**-D**' and '**-U**' options on the command line before processing '**-imacros***file*', regardless of the order in which they are written. All the '**-include**' and '**-imacros**' options are processed in the order in which they are written.

-idirafter *dir* — Add the directory *dir* to the second include path. The directories on the second include path are searched when a header file is not found in any of the directories in the main include path (the one that '**-I**' adds to).

-iprefix *prefix* — Specify *prefix* as the prefix for subsequent '**-iwithprefix**' options.

-iwithprefix *dir*

Add a directory to the second include path. The directory's name is made by concatenating *prefix* and *dir*, where *prefix* was specified previously with '**-iprefix**'.

-nostdinc — Do not search the standard system directories for header files. Only the directories you have specified with '**-I**' options (and the current directory, if appropriate) are searched.

By using both '**-nostdinc**' and '**-I-**', you can limit the include-file search file to only those directories you specify explicitly.

-nostdinc++ — Do not search for header files in the C++**-specific** standard directories, but do still search the other standard directories. (This option is used when building '`libg++`'.)

-undef — Do not predefine any nonstandard macros. (Including architecture flags).

-E — Run only the C preprocessor. Preprocess all the C source files specified and output the results to standard output or to the specified output file.

-C — Tell the preprocessor not to discard comments. Used with the '**-E**' option.

-P Tell the preprocessor not to generate '`#line`' commands. Used with the '**-E**' option.

-M [**-MG**] Tell the preprocessor to output a rule suitable for `make` describing the dependencies of each object file. For each source file, the preprocessor outputs one `make`-rule whose target is the object file name for that source file and whose dependencies are all the files included by it with '`#include`'. This rule may be a single line or may be continued with '\'-newline if it is long. The list of rules is printed on standard output instead of the preprocessed C program.

'**-M**' implies '**-E**'.

'**-MG**' says to treat missing header files as generated files and assume they live in the same directory as the source file. It must be specified in addition to '**-M**'.

-MM [**-MG**] Like '**-M**' but the output mentions only the user header files included with '`#include "`*file*`"`'. System header files included with '`#include <`*file*`>`' are omitted.

-MD Like '**-M**' but the dependency information is written to files with names made by replacing '`.o`' with '`.d`' at the end of the output file names. This is in addition to compiling the file as specified—'**-MD**' does not inhibit ordinary compilation the way '**-M**' does.

The Mach utility '`md`' can be used to merge the '`.d`' files into a single dependency file suitable for using with the '`make`' command.

-MMD Like '**-MD**' except mention only user header files, not system header files.

-H Print the name of each header file used, in addition to other normal activities.

-A*question*`(`*answer*`)`
Assert the answer *answer* for *question*, in case it is tested with a preprocessor conditional such as '`#if #`*question*`(`*answer*`)`'. '**-A-**' disables the standard assertions that normally describe the target machine.

-A*question*	(*answer*) Assert the answer *answer* for *question*, in case it is tested with a preprocessor conditional such as '`#if` #*question*(*answer*)'. '**-A-**' disables the standard assertions that normally describe the target machine.
-D*macro*	Define macro *macro* with the string '`1`' as its definition.
-D*macro*=*defn*	Define macro *macro* as *defn*. All instances of '**-D**' on the command line are processed before any '**-U**' options.
-U*macro*	Undefine macro *macro*. '**-U**' options are evaluated after all '**-D**' options, but before any '**-include**' and '**-imacros**' options.
-dM	Tell the preprocessor to output only a list of the macro definitions that are in effect at the end of preprocessing. Used with the '**-E**' option.
-dD	Tell the preprocessor to pass all macro definitions into the output, in their proper sequence in the rest of the output.
-dN	Like '**-dD**' except that the macro arguments and contents are omitted. Only '`#define` *name*' is included in the output.

Assembler Options

-Wa,*option*	Pass *option* as an option to the assembler. If *option* contains commas, it is split into multiple options at the commas.

Linker Options

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

object-file-name	A file name that does not end in a special recognized suffix is considered to name an object file or library. (Object files are distinguished from libraries by the linker according to the file contents.) If GCC does a link step, these object files are used as input to the linker.

-l*library* Use the library named *library* when linking.

The linker searches a standard list of directories for the library, which is actually a file named '`lib`*library*.a'. The linker then uses this file as if it had been specified precisely by name.

The directories searched include several standard system directories plus any that you specify with '**-L**'.

Normally the files found this way are library files—archive files whose members are object files. The linker handles an archive file by scanning through it for members which define symbols that have so far been referenced but not defined. However, if the linker finds an ordinary object file rather than a library, the object file is linked in the usual fashion. The only difference between using an '**-l**' option and specifying a file name is that '**-l**' surrounds *library* with '`lib`' and '`.a`' and searches several directories.

-lobjc You need this special case of the **-l** option in order to link an Objective C program.

-nostartfiles Do not use the standard system startup files when linking. The standard libraries are used normally.

-nostdlib Don't use the standard system libraries and startup files when linking. Only the files you specify will be passed to the linker.

-static On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.

-shared Produce a shared object which can then be linked with other objects to form an executable. Only a few systems support this option.

-symbolic Bind references to global symbols when building a shared object. Warn about any unresolved references (unless overridden by the link editor option '**-Xlinker -z -Xlinker** `defs`'). Only a few systems support this option.

–Xlinker *option* — Pass *option* as an option to the linker. You can use this to supply system-specific linker options which GNU CC does not know how to recognize.

If you want to pass an option that takes an argument, you must use '**–Xlinker**' twice, once for the option and once for the argument. For example, to pass '**–assert** definitions', you must write '**–Xlinker –assert –Xlinker** `definitions`'. It does not work to write '`–Xlinker "–assert definitions"`', because this passes the entire string as a single argument, which is not what the linker expects.

–Wl, *option* — Pass *option* as an option to the linker. If *option* contains commas, it is split into multiple options at the commas.

–u *symbol* — Pretend the symbol *symbol* is undefined, to force linking of library modules to define it. You can use '**–u**' multiple times with different symbols to force loading of additional library modules.

Directory Options

These options specify directories to search for header files, for libraries and for parts of the compiler:

–I*dir* — Append directory *dir* to the list of directories searched for include files.

–I– — Any directories you specify with '**–I**' options before the '**–I–**' option are searched only for the case of '`#include "`*file*`"`'; they are not searched for '`#include <`*file*`>`'.

If additional directories are specified with '**–I**' options after the '**–I–**', these directories are searched for all '`#include`' directives. (Ordinarily *all* '**–I**' directories are used this way.)

In addition, the '**–I–**' option inhibits the use of the current directory (where the current input file came from) as the first search directory for '`#include "`*file*`"`'. There is no way to override this effect of '**–I–**'. With '**–I.**' you can specify searching the directory which was current when the compiler was invoked. That is not exactly the same as what the preprocessor does by default, but it is often satisfactory.

'**-I-**' does not inhibit the use of the standard system directories for header files. Thus, '**-I-**' and '**-nostdinc**' are independent.

-L*dir* Add directory *dir* to the list of directories to be searched for '**-l**'.

-B*prefix* This option specifies where to find the executables, libraries and data files of the compiler itself.

The compiler driver program runs one or more of the subprograms '`cpp`', '`cc1`' (or, for C++, '`cc1plus`'), '`as`' and '`ld`'. It tries *prefix* as a prefix for each program it tries to run, both with and without '*machine*/*version*/'.

For each subprogram to be run, the compiler driver first tries the '**-B**' prefix, if any. If that name is not found, or if '**-B**' was not specified, the driver tries two standard prefixes, which are '`/usr/lib/gcc/`' and '`/usr/local/lib/gcc-lib/`'. If neither of those results in a file name that is found, the compiler driver searches for the unmodified program name, using the directories specified in your '`PATH`' environment variable.

The run-time support file '`libgcc.a`' is also searched for using the '**-B**' prefix, if needed. If it is not found there, the two standard prefixes above are tried, and that is all. The file is left out of the link if it is not found by those means. Most of the time, on most machines, '`libgcc.a`' is not actually necessary.

You can get a similar result from the environment variable `GCC_EXEC_PREFIX`; if it is defined, its value is used as a prefix in the same way. If both the '**-B**' option and the `GCC_EXEC_PREFIX` variable are present, the '**-B**' option is used first and the environment variable value second.

Warning Options

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there may have been an error.

These options control the amount and kinds of warnings produced by GNU CC:

-fsyntax-only Check the code for syntax errors, but don't emit any output.

-w Inhibit all warning messages.

-Wno-import Inhibit warning messages about the use of `#import`.

-pedantic Issue all the warnings demanded by strict ANSI standard C; reject all programs that use forbidden extensions.

Valid ANSI standard C programs should compile properly with or without this option (though a rare few will require '**-ansi**'). However, without this option, certain GNU extensions and traditional C features are supported as well. With this option, they are rejected. There is no reason to *use* this option; it exists only to satisfy pedants.

'**-pedantic**' does not cause warning messages for use of the alternate keywords whose names begin and end with '__'. Pedantic warnings are also disabled in the expression that follows `__extension__`. However, only system header files should use these escape routes; application programs should avoid them.

-pedantic-errors
Like '**-pedantic**', except that errors are produced rather than warnings.

-W Print extra warning messages for these events:

1) A nonvolatile automatic variable might be changed by a call to `longjmp`. These warnings are possible only in optimizing compilation.

The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because `longjmp` cannot in fact be called at the place which would cause a problem.

2) A function can return either with or without a value. (Falling off the end of the function body is considered returning without a value.) For example, this function would evoke such a warning:

```
foo (a) {   if (a > 0)     return a; }
```

Spurious warnings can occur because GNU CC does not realize that certain functions (including `abort` and `longjmp`) will never return.

3) An expression-statement or the left-hand side of a comma expression contains no side effects. To suppress the warning, cast the unused expression to void. For example, an expression such as '`x[i,j]`' will cause a warning, but '`x[(void)i,j]`' will not.

4) An unsigned value is compared against zero with '>' or '<='.

-Wimplicit Warn whenever a function or parameter is implicitly declared.

-Wreturn-type Warn whenever a function is defined with a return-type that defaults to `int`. Also warn about any `return` statement with no return-value in a function whose return-type is not `void`.

-Wunused Warn whenever a local variable is unused aside from its declaration, whenever a function is declared static but never defined, and whenever a statement computes a result that is explicitly not used.

-Wswitch Warn whenever a `switch` statement has an index of enumeral type and lacks a `case` for one or more of the named codes of that enumeration. (The presence of a `default` label prevents this warning.) `case` labels outside the enumeration range also provoke warnings when this option is used.

-Wcomment Warn whenever a comment-start sequence `/*` appears in a comment.

-Wtrigraphs Warn if any trigraphs are encountered (assuming they are enabled).

-Wformat Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified.

-Wchar-subscripts
Warn if an array subscript has type `char`. This is a common cause of error, as programmers often forget that this type is signed on some machines.

-Wuninitialized
An automatic variable is used without first being initialized.

These warnings are possible only in optimizing compilation, because they require data flow information that is computed only when optimizing. If you don't specify '**-O**', you simply won't get these warnings.

These warnings occur only for variables that are candidates for register allocation. Therefore, they do not occur for a variable that is declared `volatile`, or whose address is taken, or whose size is other than 1, 2, 4 or 8 bytes. Also, they do not occur for structures, unions or arrays, even when they are in registers.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

These warnings are made optional because GNU CC is not smart enough to see all the reasons why the code might be correct despite appearing to have an error. Here is one example of how this can happen:

```
{   int x;   switch (y)
  { case 1: x = 1;      break;
    case 2: x = 4;       break;
    case 3: x = 5;     }
 foo (x); }
```

If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but GNU CC doesn't know this. Here is another common case:

```
{   int save_y;
    if (change_y) save_y = y, y = new_y; ...
    if (change_y) y = save_y; }
```

This has no bug because `save_y` is used only if it is set.

Some spurious warnings can be avoided if you declare as `volatile` all the functions you use that never return.

-Wparentheses Warn if parentheses are omitted in certain contexts.

-Wtemplate-debugging
When using templates in a C++ program, warn if debugging is not yet fully available (C++ only).

-Wall All of the above '**-W**' options combined. These are all the options which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

The remaining **-W**... options are not implied by **-Wall** because they warn about constructions that we consider reasonable to use, on occasion, in clean programs.

-Wtraditional Warn about certain constructs that behave differently in traditional and ANSI C.

1) Macro arguments occurring within string constants in the macro body. These would substitute the argument in traditional C, but are part of the constant in ANSI C.

2) A function declared external in one block and then used after the end of the block.

3) A `switch` statement has an operand of type `long`.

-Wshadow Warn whenever a local variable shadows another local variable.

-Wid-clash-*len* Warn whenever two distinct identifiers match in the first *len* characters. This may help you prepare a program that will compile with certain obsolete, brain-damaged compilers.

-Wpointer-arith Warn about anything that depends on the *size* of a function type or of `void`. GNU C assigns these types a size of 1, for convenience in calculations with `void *` pointers and pointers to functions.

-Wcast-qual Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a `const char *` is cast to an ordinary `char *`.

-Wcast-align Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a `char *` is cast to an `int *` on machines where integers can only be accessed at two- or four-byte boundaries.

-Wwrite-strings Give string constants the type `const char[`*length*`]` so that copying the address of one into a non-`const char *` pointer will get a warning. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it will just be a nuisance; this is why we did not make **-Wall** request these warnings.

-Wconversion Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed point argument except when the same as the default promotion.

-Waggregate-return
Warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning.)

-Wstrict-prototypes
Warn if a function is declared or defined without specifying the argument types. (An old-style function definition is permitted without a warning if preceded by a declaration which specifies the argument types.)

-Wmissing-prototypes
Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. The aim is to detect global functions that fail to be declared in header files.

-Wmissing-declarations
Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files.

-Wredundant-decls
Warn if anything is declared more than once in the same scope, even in cases where multiple declaration is valid and changes nothing.

-Wnested-externs
Warn if an `extern` declaration is encountered within an function.

-Wenum-clash Warn about conversion between different enumeration types (C++ only).

-Woverloaded-virtual
(C++ only.) In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. Use this option to request warnings when a derived class declares a function that may be an erroneous attempt to define a virtual function: that is, warn when a function with the

same name as a virtual function in the base class, but with a type signature that doesn't match any virtual functions from the base class.

-Winline Warn if a function can not be inlined, and either it was declared as inline, or else the **-finline-functions** option was given.

-Werror Treat warnings as errors; abort compilation after any warning.

Debugging Options

GNU CC has various special options that are used for debugging either your program or GCC:

-g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

On most systems that use stabs format, '**-g**' enables use of extra debugging information that only GDB can use; this extra information makes debugging work better in GDB but will probably make other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use '**-gstabs+**', '**-gstabs**', '**-gxcoff+**', '**-gxcoff**', '**-gdwarf+**', or '**-gdwarf**' (see below).

Unlike most other C compilers, GNU CC allows you to use '**-g**' with '**-O**'. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

The following options are useful when GNU CC is generated with the capability for more than one debugging format.

-ggdb Produce debugging information in the native format (if that is supported), including GDB extensions if at all possible.

-gstabs Produce debugging information in stabs format (if that is supported), without GDB extensions. This is the format used by DBX on most BSD systems.

-gstabs+ Produce debugging information in stabs format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

-gcoff Produce debugging information in COFF format (if that is supported). This is the format used by BSD on most System V systems prior to System V Release 4.

-gxcoff Produce debugging information in XCOFF format (if that is supported). This is the format used by the DBX debugger on IBM RS/6000 systems.

-gxcoff+ Produce debugging information in XCOFF format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

-g*level*
-ggdb*level*
-gstabs*level*
-gcoff*level*
-gxcoff*level* Request debugging information and also use *level* to specify how much information. The default level is 2.

Level 1 produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers.

Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use '**-g3**'.

-p Generate extra code to write profile information suitable for the analysis program `prof`.

-pg Generate extra code to write profile information suitable for the analysis program `gprof`.

-a Generate extra code to write profile information for basic blocks, which will record the number of times each basic block is executed. This data could be analyzed by a program like `tcov`. Note, however, that the format of the data is not what `tcov` expects. Eventually GNU `gprof` should be extended to process this data.

-d*letters* Says to make debugging dumps during compilation at times specified by *letters*. This is used for debugging the compiler. The file names for most of the dumps are made by appending a word to the source file name (e.g. '`foo.c.rtl`' or '`foo.c.jump`').

-dM Dump all macro definitions, at the end of preprocessing, and write no output.

-dN Dump all macro names, at the end of preprocessing.

-dD Dump all macro definitions, at the end of preprocessing, in addition to normal output.

-dy Dump debugging information during parsing, to standard error.

-dr Dump after RTL generation, to '*file*`.rtl`'.

-dx Just generate RTL for a function instead of compiling it. Usually used with '`r`'.

-dj Dump after first jump optimization, to '*file*`.jump`'.

-ds Dump after CSE (including the jump optimization that sometimes follows CSE), to '*file*`.cse`'.

-dL	Dump after loop optimization, to '*file*.`loop`'.
-dt	Dump after the second CSE pass (including the jump optimization that sometimes follows CSE), to '*file*.`cse2`'.
-df	Dump after flow analysis, to '*file*.`flow`'.
-dc	Dump after instruction combination, to '*file*.`combine`'.
-dS	Dump after the first instruction scheduling pass, to '*file*.`sched`'.
-dl	Dump after local register allocation, to '*file*.`lreg`'.
-dg	Dump after global register allocation, to '*file*.`greg`'.
-dR	Dump after the second instruction scheduling pass, to '*file*.`sched2`'.
-dJ	Dump after last jump optimization, to '*file*.`jump2`'.
-dd	Dump after delayed branch scheduling, to '*file*.`dbr`'.
-dk	Dump after conversion from registers to stack, to '*file*.`stack`'.
-da	Produce all the dumps listed above.
-dm	Print statistics on memory usage, at the end of the run, to standard error.
-dp	Annotate the assembler output with a comment indicating which pattern and alternative was used.

-fpretend-float

When running a cross-compiler, pretend that the target machine uses the same floating point format as the host machine. This causes incorrect output of the actual floating constants, but the actual instruction sequence will probably be the same as GNU CC would make when running on the target machine.

-save-temps Store the usual temporary intermediate files permanently; place them in the current directory and name them based on the source file. Thus, compiling '`foo.c`' with '**-c -save-temps**' would produce files '`foo.cpp`' and '`foo.s`', as well as '`foo.o`'.

-print-file-name=*library*
Print the full absolute name of the library file *library* that would be used when linking—and do not do anything else. With this option, GNU CC does not compile or link anything; it just prints the file name.

-print-libgcc-file-name
Same as '**-print-file-name=libgcc.a**'.

-print-prog-name=*program*
Like '**-print-file-name**', but searches for a program such as 'cpp'.

Optimization Options

These options control various sorts of optimizations:

-O
-O1
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

Without '**-O**', the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Without '**-O**', only variables declared `register` are allocated in registers. The resulting compiled code is a little worse than produced by PCC without '**-O**'.

With '**-O**', the compiler tries to reduce code size and execution time.

When you specify '**-O**', the two options '**-fthread-jumps**' and '**-fdefer-pop**' are turned on. On machines that have delay slots, the '**-fdelayed-branch**' option is turned on. For those machines that can support debugging even without a frame pointer, the '**-fomit-frame-pointer**' option is turned on. On some machines other flags may also be turned on.

-O2 Optimize even more. Nearly all supported optimizations that do not involve a space-speed tradeoff are performed. Loop unrolling and function inlining are not done, for example. As compared to **-O**, this option increases both compilation time and the performance of the generated code.

-O3 Optimize yet more. This turns on everything **-O2** does, along with also turning on **-finline-functions.**

-O0 Do not optimize.

If you use multiple **-O** options, with or without level numbers, the last such option is the one that is effective.

Options of the form '**-f***flag*' specify machine-independent flags. Most flags have both positive and negative forms; the negative form of '**-ffoo**' would be '**-fno-foo**'. The following list shows only one form—the one which is not the default. You can figure out the other form by either removing 'no-' or adding it.

-ffloat-store Do not store floating point variables in registers. This prevents undesirable excess precision on machines such as the 68000 where the floating registers (of the 68881) keep more precision than a double is supposed to have.

For most programs, the excess precision does only good, but a few programs rely on the precise definition of floating point. Use '**-ffloat-store**' for such programs.

-fmemoize-lookups
-fsave-memoized
Use heuristics to compile faster (C++ only). These heuristics are not enabled by default, since they are only effective for certain input files. Other input files compile more slowly.

The first time the compiler must build a call to a member function (or reference to a data member), it must (1) determine whether the class implements member functions of that name; (2) resolve which member function to call (which involves figuring out what sorts of type conversions need to be made); and (3) check the visibility of the member function to the caller. All of this adds up to slower compilation. Normally, the second time a call is made to that member function (or reference to that data member), it must go through the same lengthy process again. This means that code like this

```
cout << "This " << p << " has " << n << " legs.\n";
```

makes six passes through all three steps. By using a software cache, a hit significantly reduces this cost. Unfortunately, using the cache introduces another layer of mechanisms which must be implemented, and so incurs its own overhead. '**-fmemoize-lookups**' enables the software cache.

Because access privileges (visibility) to members and member functions may differ from one function context to the next, `g++` may need to flush the cache. With the '**-fmemoize-lookups**' flag, the cache is flushed after every function that is compiled. The '**-fsave-memoized**' flag enables the same software cache, but when the compiler determines that the context of the last function compiled would yield the same access privileges of the next function to compile, it preserves the cache. This is most helpful when defining many member functions for the same class: with the exception of member functions which are friends of other classes, each member function has exactly the same access privileges as every other, and the cache need not be flushed.

-fno-default-inline

Don't make member functions inline by default merely because they are defined inside the class scope (C++ only).

-fno-defer-pop Always pop the arguments to each function call as soon as that function returns. For machines which must pop arguments after a function call, the compiler normally lets arguments accumulate on the stack for several function calls and pops them all at once.

-fforce-mem Force memory operands to be copied into registers before doing arithmetic on them. This may produce better code by making all memory references potential common subexpressions. When they are not common subexpressions, instruction combination should eliminate the separate register-load. I am interested in hearing about the difference this makes.

-fforce-addr Force memory address constants to be copied into registers before doing arithmetic on them. This may produce better code just as '**-fforce-mem**' may. I am interested in hearing about the difference this makes.

-fomit-frame-pointer

Don't keep the frame pointer in a register for functions that don't need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions. *It also makes debugging impossible on most machines.*

On some machines, such as the Vax, this flag has no effect, because the standard calling sequence automatically handles the frame pointer and nothing is saved by pretending it doesn't exist. The machine-description macro `FRAME_POINTER_REQUIRED` controls whether a target machine supports this flag.

-finline-functions

Integrate all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way.

If all calls to a given function are integrated, and the function is declared `static`, then GCC normally does not output the function as assembler code in its own right.

-fcaller-saves Enable values to be allocated in registers that will be clobbered by function calls, by emitting extra instructions to save and restore the registers around such calls. Such allocation is done only when it seems to result in better code than would otherwise be produced.

This option is enabled by default on certain machines, usually those which have no call-preserved registers to use instead.

-fkeep-inline-functions
Even if all calls to a given function are integrated, and the function is declared `static`, nevertheless output a separate run-time callable version of the function.

-fno-function-cse
Do not put function addresses in registers; make each instruction that calls a constant function contain the function's address explicitly.

This option results in less efficient code, but some strange hacks that alter the assembler output may be confused by the optimizations performed when this option is not used.

-fno-peephole Disable any machine-specific peephole optimizations.

-ffast-math This option allows GCC to violate some ANSI or rules/specifications in the interest of optimizing code for speed. For example, it allows the compiler to assume arguments to the `sqrt` function are non-negative numbers.

This option should never be turned on by any '**-O**' option since it can result in incorrect output for programs which depend on an exact implementation of or ANSI rules/specifications for math functions.

The following options control specific optimizations. The '**-O2**' option turns on all of these optimizations except '**-funroll-loops**' and '**-funroll-all-loops**'.

The '**-O**' option usually turns on the '**-fthread-jumps**' and '**-fdelayed-branch**' options, but specific machines may change the default optimizations.

You can use the following flags in the rare cases when fine-tuning of optimizations to be performed is desired.

-fstrength-reduce
Perform the optimizations of loop strength reduction and elimination of iteration variables.

-fthread-jumps Perform optimizations where we check to see if a jump branches to a location where another comparison subsumed by the first is found. If so, the first branch is redirected to either the destination of the second branch or a point immediately following it, depending on whether the condition is known to be true or false.

-funroll-loops Perform the optimization of loop unrolling. This is only done for loops whose number of iterations can be determined at compile time or run time.

-funroll-all-loops
Perform the optimization of loop unrolling. This is done for all loops. This usually makes programs run more slowly.

-fcse-follow-jumps
In common subexpression elimination, scan through jump instructions when the target of the jump is not reached by any other path. For example, when CSE encounters an `if` statement with an `else` clause, CSE will follow the jump when the condition tested is false.

-fcse-skip-blocks
This is similar to '**-fcse-follow-jumps**', but causes CSE to follow jumps which conditionally skip over blocks. When CSE encounters a simple `if` statement with no else clause, '**-fcse-skip-blocks**' causes CSE to follow the jump around the body of the `if`.

-frerun-cse-after-loop
Re-run common subexpression elimination after loop optimizations has been performed.

-felide-constructors

Elide constructors when this seems plausible (C++ only). With this flag, GNU C++ initializes `y` directly from the call to `foo` without going through a temporary in the following code:

```
A foo (); A y = foo ();
```

Without this option, GNU C++ first initializes `y` by calling the appropriate constructor for type `A`; then assigns the result of `foo` to a temporary; and, finally, replaces the initial value of '`y`' with the temporary.

The default behavior ('**-fno-elide-constructors**') is specified by the draft ANSI C++ standard. If your program's constructors have side effects, using '**-felide-constructors**' can make your program act differently, since some constructor calls may be omitted.

-fexpensive-optimizations

Perform a number of minor optimizations that are relatively expensive.

-fdelayed-branch

If supported for the target machine, attempt to reorder instructions to exploit instruction slots available after delayed branch instructions.

-fschedule-insns

If supported for the target machine, attempt to reorder instructions to eliminate execution stalls due to required data being unavailable. This helps machines that have slow floating point or memory load instructions by allowing other instructions to be issued until the result of the load or floating point instruction is required.

-fschedule-insns2

Similar to '**-fschedule-insns**', but requests an additional pass of instruction scheduling after register allocation has been done. This is especially useful on machines with a relatively small number of registers and where memory load instructions take more than one cycle.

Target Options

By default, GNU CC compiles code for the same type of machine that you are using. However, it can also be installed as a cross-compiler, to compile for some other type of machine. In fact, several different configurations of GNU CC, for different target machines, can be installed side by side. Then you specify which one to use with the '**-b**' option.

In addition, older and newer versions of GNU CC can be installed side by side. One of them (probably the newest) will be the default, but you may sometimes wish to use another.

-b *machine* The argument *machine* specifies the target machine for compilation. This is useful when you have installed GNU CC as a cross-compiler.

The value to use for *machine* is the same as was specified as the machine type when configuring GNU CC as a cross-compiler. For example, if a cross-compiler was configured with '`configure` i386v', meaning to compile for an 80386 running System V, then you would specify '`-b i386v`' to run that cross compiler.

When you do not specify '**-b**', it normally means to compile for the same type of machine that you are using.

-V *version* The argument *version* specifies which version of GNU CC to run. This is useful when multiple versions are installed. For example, *version* might be '`2.0`', meaning to run GNU CC version 2.0.

The default version, when you do not specify '**-V**', is controlled by the way GNU CC is installed. Normally, it will be a version that is recommended for general use.

Machine Dependent Options

Each of the target machine types can have its own special options, starting with '**-m**', to choose among various hardware models or configurations—for example, 68010 vs 68020, floating coprocessor or none. A single installed version of the compiler can compile for any model or configuration, according to the options specified.

Some configurations of the compiler also support additional special options, usually for command-line compatibility with other compilers on the same platform.

These '**-m**' options are defined for the 68k:

-m68000
-mc68000 Generate output for a 68000.

-m68020
-mc68020 Generate output for a 68020 (rather than a 68000). This is the default when the compiler is configured for 68020-based systems.

-m68030 Generate output for a 68030. This is the default when the compiler is configured for 68030-based systems.

-m68040 Generate output for a 68040. This is the default when the compiler is configured for 68040-based systems.

-m68020-40 Generate output for a 68040, without using any of the new instructions. This results in code which can run relatively efficiently on either a 68020/68881 or a 68030 or a 68040.

-mfpa Generate output containing Sun FPA instructions for floating point.

-msoft-float Generate output containing library calls for floating point. *WARNING:* the requisite libraries are not part of GNU CC. Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

-mshort Consider type `int` to be 16 bits wide, like `short int`.

-mnobitfield Do not use the bit-field instructions. '**-m68000**' implies '**-mnobitfield**'.

-mbitfield Do use the bit-field instructions. '**-m68020**' implies '**-mbitfield**'. This is the default if you use the unmodified sources.

-mrtd Use a different function-calling convention, in which functions that take a fixed number of arguments return with the `rtd` instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there.

This calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler.

Also, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); otherwise incorrect code will be generated for calls to those functions.

In addition, seriously incorrect code will result if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.)

The `rtd` instruction is supported by the 68010 and 68020 processors, but not by the 68000.

These '**-m**' options are defined for the SPARC:

-mfpu
-mhard-float Generate output containing floating point instructions. This is the default.

-mno-fpu
-msoft-float Generate output containing library calls for floating point. *Warning:* there is no GNU floating-point library for SPARC. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

-msoft-float changes the calling convention in the output file; therefore, it is only useful if you compile *all* of a program with this option.

-mno-epilogue

-mepilogue With **-mepilogue** (the default), the compiler always emits code for function exit at the end of each function. Any function exit in the middle of the function (such as a return statement in C) will generate a jump to the exit code at the end of the function.

With **-mno-epilogue**, the compiler tries to emit exit code inline at every function exit.

-mno-v8

-mv8

-msparc**lite** These three options select variations on the SPARC architecture.

By default (unless specifically configured for the Fujitsu SPARClite), GCC generates code for the v7 variant of the SPARC architecture.

-mv8 will give you SPARC v8 code. The only difference from v7 code is that the compiler emits the integer multiply and integer divide instructions which exist in SPARC v8 but not in SPARC v7.

-msparclite will give you SPARClite code. This adds the integer multiply, integer divide step and scan (ffs) instructions which exist in SPARClite but not in SPARC v7.

-mcypress

-msupersparc These two options select the processor for which the code is optimized.

With **-mcypress** (the default), the compiler optimizes code for the Cypress CY7C602 chip, as used in the SparcStation/SparcServer 3xx series. This is also appropriate for the older SparcStation 1, 2, IPX etc.

With **-msupersparc** the compiler optimizes code for the SuperSparc cpu, as used in the SparcStation 10, 1000 and 2000 series. This flag also enables use of the full SPARC v8 instruction set.

These '**-m**' options are defined for the PowerPC:

-mfp-in-toc
-mno-fp-in-toc
: Control whether or not floating-point constants go in the Table of Contents (TOC), a table of all global variable and function addresses. By default GCC puts floating-point constants there; if the TOC overflows, '**-mno-fp-in-toc**' will reduce the size of the TOC, which may avoid the overflow.

These '**-m**' options are defined for the x86:

-m486
-mno-486
: Control whether or not code is optimized for a 486 instead of an 386. Code generated for a 486 will run on a 386 and vice versa.

-msoft-float
: Generate output containing library calls for floating point. *Warning:* the requisite libraries are not part of GNU CC. Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

On machines where a function returns floating point results in the 80387 register stack, some floating point opcodes may be emitted even if '**-msoft-float**' is used.

-mno-fp-ret-in-387
: Do not use the FPU registers for return values of functions.

The usual calling convention has functions return values of types `float` and `double` in an FPU register, even if there is no FPU. The idea is that the operating system should emulate an FPU.

The option '**-mno-fp-ret-in-387**' causes such values to be returned in ordinary CPU registers instead.

Code Generation Options

These machine-independent options control the interface conventions used in code generation.

Most of them begin with '**-f**'. These options have both positive and negative forms; the negative form of '**-ffoo**' would be '**-fno-foo**'. In the table below, only one of the forms is listed—the one which is not the default. You can figure out the other form by either removing 'no-' or adding it.

-fnonnull-objects

Assume that objects reached through references are not null (C++ only).

Normally, GNU C++ makes conservative assumptions about objects reached through references. For example, the compiler must check that a is not null in code like the following:

```
obj &a = g (); a.f (2);
```

Checking that references of this sort have non-null values requires extra code, however, and it is unnecessary for many programs. You can use '**-fnonnull-objects**' to omit the checks for null, if your program doesn't require checking.

-fpcc-struct-return

Use the same convention for returning struct and union values that is used by the usual C compiler on your system. This convention is less efficient for small structures, and on many machines it fails to be reentrant; but it has the advantage of allowing intercallability between GCC-compiled code and PCC-compiled code.

-freg-struct-return

Use the convention that struct and union values are returned in registers when possible. This is more efficient for small structures than **-fpcc-struct-return**.

If you specify neither **-fpcc-struct-return** nor **-freg-struct-return**, GNU CC defaults to whichever convention is standard for the target. If there is no standard convention, GNU CC defaults to **-fpcc-struct-return**.

-fshort-enums Allocate to an `enum` type only as many bytes as it needs for the declared range of possible values. Specifically, the `enum` type will be equivalent to the smallest integer type which has enough room.

-fshort-double Use the same size for `double` as for `float`.

-fshared-data Requests that the data and non-`const` variables of this compilation be shared data rather than private data. The distinction makes sense only on certain operating systems, where shared data is shared between processes running the same program, while private data exists in one copy per process.

-fno-common
-fcommon The default, **-fno-common**, allocates even uninitialized global variables in the bss section of the object file, rather than generating them as common blocks. This has the effect that if the same variable is declared (without `extern`) in two different compilations, you will get an error when you link them.

-fno-ident Ignore the '`#ident`' directive.

-fno-gnu-linker Do not output global initializations (such as C++ constructors and destructors) in the form used by the GNU linker (on systems where the GNU linker is the standard method of handling them). Use this option when you want to use a non-GNU linker, which also requires using the `collect2` program to make sure the system linker includes constructors and destructors. (`collect2` is included in the GNU CC distribution.) For systems which *must* use `collect2`, the compiler driver `gcc` is configured to do this automatically.

-finhibit-size-directive
Don't output a `.size` assembler directive, or anything else that would cause trouble if the function is split in

the middle, and the two halves are placed at locations far apart in memory. This option is used when compiling '`crtstuff.c`'; you should not need to use it for anything else.

-fverbose-asm Put extra commentary information in the generated assembly code to make it more readable. This option is generally only of use to those who actually need to read the generated assembly code (perhaps while debugging the compiler itself).

-fvolatile Consider all memory references through pointers to be volatile.

-fvolatile-global
Consider all memory references to extern and global data items to be volatile.

-fpic If supported for the target machines, generate position-independent code, suitable for use in a shared library.

-fPIC If supported for the target machine, emit position-independent code, suitable for dynamic linking, even if branches need large displacements.

-ffixed-*reg* Treat the register named *reg* as a fixed register; generated code should never refer to it (except perhaps as a stack pointer, frame pointer or in some other fixed role).

reg must be the name of a register. The register names accepted are machine-specific and are defined in the `REGISTER_NAMES` macro in the machine description macro file.

This flag does not have a negative form, because it specifies a three-way choice.

-fcall-used-*reg*
Treat the register named *reg* as an allocatable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register *reg*.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

This flag does not have a negative form, because it specifies a three-way choice.

-fcall-saved-*reg*
: Treat the register named *reg* as an allocatable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register *reg* if they use it.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

A different sort of disaster will result from the use of this flag for a register in which function values may be returned.

This flag does not have a negative form, because it specifies a three-way choice.

PRAGMAS

Two '`#pragma`' directives are supported for GNU C++, to permit using the same header file for two purposes: as a definition of interfaces to a given object class, and as the full definition of the contents of that object class.

`#pragma interface`
: (C++ only.) Use this directive in header files that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing '`#pragma interface`' is included in a

compilation, this auxiliary information will not be generated (unless the main input source file itself uses '`#pragma implementation`'). Instead, the object files will contain references to be resolved at link time.

`#pragma implementation`
`#pragma implementation "`*objects*`.h"`

(C++ only.) Use this pragma in a main input file, when you want full output from included header files to be generated (and made globally visible). The included header file, in turn, should use '`#pragma interface`'. Backup copies of inline member functions, debugging information, and the internal tables used to implement virtual functions are all generated in implementation files.

If you use '`#pragma implementation`' with no argument, it applies to an include file with the same basename as your source file; for example, in '`allclass.cc`', '`#pragma implementation`' by itself is equivalent to '`#pragma implementation "allclass.h"`'. Use the string argument if you want a single implementation file to include code from multiple header files.

There is no way to split up the contents of a single header file into multiple implementation files.

FILES

`file.c`	C source file
`file.h`	C header (preprocessor) file
`file.i`	preprocessed C source file
`file.C`	C++ source file
`file.cc`	C++ source file
`file.cxx`	C++ source file

`file.m`	Objective-C source file
`file.s`	assembly language file
`file.o`	object file
`a.out`	link edited output
TMPDIR/`cc*`	temporary files
LIBDIR/`cpp`	preprocessor
LIBDIR/`cc1`	compiler for C
LIBDIR/`cc1plus`	compiler for C++
LIBDIR/`collect`	linker front end needed on some machines
LIBDIR/`libgcc.a`	GCC subroutine library
`/lib/crt[01n].o`	start-up routine
LIBDIR/`ccrt0`	additional start-up routine for C++
`/lib/libc.a`	standard C library, see *intro*(3)
`/usr/include`	standard directory for `#include` files
LIBDIR/`include`	standard `gcc` directory for `#include` files
LIBDIR/`g++-include`	additional g++ directory for `#include`

Note that *LIBDIR* is usually `/usr/local/lib/`*machine*/*version*.

TMPDIR comes from the environment variable `TMPDIR` (default `/usr/tmp` if available, else `/tmp`).

SEE ALSO

Utility Programs - `cpp, as, ld, gdb, adb, dbx, sdb`

Documents - *Using and Porting GNU CC (for version 2.0)*, Richard M. Stallman;

The C Preprocessor, Richard M. Stallman;

Debugging with GDB: the GNU Source-Level Debugger, Richard M. Stallman and Roland H. Pesch;

Using as: the GNU Assembler, Dean Elsner, Jay Fenlason & friends;

ld: the GNU linker, Steve Chamberlain and Roland Pesch.

NOTES

The GNU C and C++ compilers (`gcc`, `g++`) are the default compilers on all LynxOS platforms.

If you have applications from previous LynxOS releases that use the proprietary Lynx C-style in-line assembly (x86 and 68k only), please see the `l-cc(1)` manual page.

The GNU C and C++ compilers are able to recognize the environment variable `ENV_PREFIX`. When `ENV_PREFIX` is set (and exported), `gcc/g++` will look in:

- `${ENV_PREFIX}/usr/include` for header files
- `${ENV_PREFIX}/lib` for archives
- `${ENV_PREFIX}/usr/lib/gcc-lib` for GNU libraries and tools.

`ENV_PREFIX` is usually set by `SETUP.*sh`, an environment setup script used for cross development with LynxOS.

COPYING

AUTHORS

See the GNU CC manual for the contributors to GNU CC.

NAME

`gdb` – The GNU Debugger

SYNOPSIS

gdb [-help] [-nx] [-q] [-batch] [-cd=*dir***] [-f] [-b** *bps***]**
[-tty=*dev***] [-s** *symfile***] [-e** *prog***] [-se** *prog***] [-c** *core***]**
[-x *cmds***] [-d** *dir***] [** *prog***[** *core***|***procID***]]**

DESCRIPTION

The purpose of a debugger such as GDB is to allow you to see what is going on “inside” another program while it executes—or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

You can use GDB to debug programs written in C, C++, and Modula-2. Fortran support will be added when a GNU Fortran compiler is ready.

GDB is invoked with the shell command `gdb`. Once started, it reads commands from the terminal until you tell it to exit with the GDB command `quit`. You can get online help from `gdb` itself by using the command `help`.

You can run `gdb` with no arguments or options; but the most usual way to start GDB is with one argument or two, specifying an executable program as the argument:

```
gdb program
```

You can also start with both an executable program and a core file specified:

```
gdb program core
```

You can, instead, specify a process ID as a second argument, if you want to debug a running process:

```
gdb program 1234
```

would attach GDB to process `1234` (unless you also have a file named '`1234`'; GDB does check for a core file first).

Here are some of the most frequently needed GDB commands:

`break` [*file*`:`]*function*
: Set a breakpoint at *function* (in *file*).

`run` [*arglist*]
: Start your program (with *arglist*, if specified).

`bt`
: Backtrace: display the program stack.

`print` *expr*
: Display the value of an expression.

`c`
: Continue running your program (after stopping, e.g. at a breakpoint).

`next`
: Execute next program line (after stopping); step *over* any function calls in the line.

`step`
: Execute next program line (after stopping); step *into* any function calls in the line.

`help` [*name*]
: Show information about GDB command *name*, or general information about using GDB.

`quit`
: Exit from GDB.

For full details on GDB, see *Using* GDB*: A Guide to the* GNU *Source-Level Debugger*, by Richard M. Stallman and Roland H. Pesch.

OPTIONS

Any arguments other than options specify an executable file and core file (or process ID); that is, the first argument encountered with no associated option flag is equivalent to a '**-se**' option, and the second, if any, is equivalent to a '**-c**' option if it's the name of a file. Many options have both long and short forms; both are shown here. The long forms are also recognized if you truncate them, so long as enough of the option is present to be unambiguous. (If you prefer, you can flag option arguments with '+' rather than '–', though we illustrate the more usual convention.)

All the options and command line arguments you give are processed in sequential order. The order makes a difference when the '**-x**' option is used.

-help
-h List all options, with brief explanations.

-symbols=*file*
-s *file* Read symbol table from file *file*.

-exec=*file*
-e *file* Use file *file* as the executable file to execute when appropriate, and for examining pure data in conjunction with a core dump.

-se=*file* Read symbol table from file *file* and use it as the executable file.

-core=*file*
-c *file* Use file *file* as a core dump to examine.

-command=*file*
-x *file* Execute GDB commands from file *file*.

-directory=*directory*
-d *directory* Add *directory* to the path to search for source files.

-nx
-n Do not execute commands from any '`.gdbinit`' initialization files. Normally, the commands in these files are executed after all the command options and arguments have been processed.

-quiet
-q "Quiet". Do not print the introductory and copyright messages. These messages are also suppressed in batch mode.

-batch Run in batch mode. Exit with status `0` after processing all the command files specified with '**-x**' (and '`.gdbinit`', if not inhibited). Exit with nonzero status if an error occurs in executing the GDB commands in the command files.

Batch mode may be useful for running GDB as a filter, for example to download and run a program on another computer; in order to make this more useful, the message

```
Program exited normally.
```

(which is ordinarily issued whenever a program running under GDB control terminates) is not issued when running in batch mode.

-cd=*directory* Run GDB using *directory* as its working directory, instead of the current directory.

-fullname
-f Emacs sets this option when it runs GDB as a subprocess. It tells GDB to output the full file name and line number in a standard, recognizable fashion each time a stack frame is displayed (which includes each time the program stops). This recognizable format looks like two '` 32`' characters, followed by the file name, line number and character position separated by colons, and a newline. The Emacs-to-GDB interface program uses the two '` 32`' characters as a signal to display the source code for the frame.

-b *bps* Set the line speed (baud rate or bits per second) of any serial interface used by GDB for remote debugging.

-tty=*device*	Run using *device* for your program's standard input and output.

SEE ALSO

Documents - *Using GDB: A Guide to the GNU Source-Level Debugger*, Richard M. Stallman and Roland H. Pesch, July 1991.

COPYING

NAME

`gperf` – generate a perfect hash function from a key set

SYNOPSIS

`gperf [ -adghijklnoprsStv ] [` *keyfile* **`]`**

DESCRIPTION

`gperf` reads a set of “keys” from *keyfile* (or, by default, from the standard input) and attempts to find a non-minimal perfect hashing function that recognizes a member of the key set in constant, i.e., O(1), time. If such a function is found the program generates a pair of *C* source code routines that perform the hashing and table lookup. All generated code is directed to the standard output.

NAME

`gprof` – display call graph profile data

SYNOPSIS

gprof [-abcsz] [-e|-E *name* **] [-f|-F** *name* **] [-k** *fromname toname* **] [** *objfile* **[** *gmon.out* **]]**

DESCRIPTION

`gprof` produces an execution profile of C, Pascal, or Fortran77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file ('`gmon.out`' default) which is created by programs that are compiled with the **-pg** option of `cc(1)`, `pc(1)`, and `f77(1)`. The **-pg** option also links in versions of the library routines that are compiled for profiling. `gprof` reads the given object file (the default is 'a.out') and establishes the relation between its symbol table and the call graph profile from '`gmon.out`'. If more than one profile file is specified, the `gprof` output shows the sum of the profile information in the given profile files.

`gprof` calculates the amount of time spent in each routine. Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. The first listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Second, a flat profile is given. This listing gives the total execution times, the call counts, the time in milliseconds the call spent in the routine itself, and the time in milliseconds the call spent in the routine itself including its descendents.

Finally, an index of the function names is provided.

OPTIONS

The following options are available:

-a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (e.g., time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *objfile* file.

-b suppresses the printing of a description of each field in the profile.

-c the static call graph of the program is discovered by a heuristic that examines the text space of the object file. Static-only parents or children are shown with call counts of 0.

-e *name* suppresses the printing of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **-e** option may be given. Only one *name* may be given with each **-e** option.

-E *name* suppresses the printing of the graph profile entry for routine *name* (and its descendants) as **-e**, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **-E** *mcount* **-E** *mcleanup* is the default.)

-f *name* prints the graph profile entry of only the specified routine *name* and its descendants. More than one **-f** option may be given. Only one *name* may be given with each **-f** option.

-F *name* prints the graph profile entry of only the routine *name* and its descendants (as **-f**) and also uses only the times of the printed routines in total time and percentage computations. More than one **-F** option may be given. Only one *name* may be given with each **-F** option. The **-F** option overrides the **-E** option.

-k *fromname toname*
will delete any arcs from routine *fromname* to routine *toname*. This can be used to break undesired cycles. More than one **-k** option may be given. Only one pair of routine names may be given with each **-k** option.

-s
a profile file 'gmon.sum' is produced that represents the sum of the profile information in all the specified profile files. This summary profile file may be given to later executions of gprof (probably also with a **-s**) to accumulate profile data across several runs of an *objfile* file.

-v
prints the version number for gprof, and then exits.

-z
displays routines that have zero usage (as shown by call counts and accumulated time). This is useful with the **-c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space.
gmon.out	dynamic call graph and profile.
gmon.sum	summarized dynamic call graph and profile.

SEE ALSO

Utility Programs -	cc
System Calls -	profil()
Library Functions -	monitor()
Articles -	"An Execution Profiler for Modular Programs", by S. Graham, P. Kessler, M. McKusick; *Software* –

Practice and Experience, Vol. 13, pp. 671-685, 1983.

"gprof: A Call Graph Execution Profiler", by S. Graham, P. Kessler, M. McKusick; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No 6, pp. 120-126, June 1982.

HISTORY

`gprof` appeared in 4.2 BSD.

BUGS

The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to the function's parents is directly proportional to the number of times that arc is traversed.

Parents that are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call `exit(2)` or return normally for the profiling information to be saved in the 'gmon.out' file.

NAME

`grep, egrep` – print lines matching a regular expression

SYNOPSIS

grep [-CVbchilnsvwx] [-*num***] [-AB** *num***] [[-e]** *expr* **| -f** *file***]**
[*files ...***]**

egrep [-CVbchilnsvwx] [-*num***] [-AB** *num***] [[-e]** *expr* **| -f** *file***]**
[*files ...***]**

DESCRIPTION

`grep` searches the files listed in the arguments (or standard input if no files are given) for all lines that contain a match for the given *expr*. If any lines match, they are printed.

Also, if any matches were found, `grep` exits with a status of 0, but if no matches were found it exits with a status of 1. This is useful for building shell scripts that use `grep` as a condition for, for example, the `if` statement.

When invoked as `egrep` the syntax of the *expr* is slightly different as expressed in the table “Grep & Egrep Expressions”.

REGULAR EXPRESSIONS

Table 1: Grep & Egrep Expressions

<table>
<tr><th>grep</th><th>egrep</th><th>Expression Explanation</th></tr>
<tr><td><code>c</code></td><td><code>c</code></td><td>A single (non-meta) character matches itself</td></tr>
<tr><td><code>.</code></td><td><code>.</code></td><td>Matches any single character except newline</td></tr>
<tr><td><code>\?</code></td><td><code>?</code></td><td>postfix operator; preceding item is optional</td></tr>
<tr><td><code>*</code></td><td><code>*</code></td><td>postfix operator; preceding item 0 or more times</td></tr>
<tr><td><code>\+</code></td><td><code>+</code></td><td>postfix operator; preceding item 1 or more times</td></tr>
<tr><td><code>\|</code></td><td><code>|</code></td><td><code>infix</code> operator; matches either argument</td></tr>
<tr><td><code>^</code></td><td><code>^</code></td><td>Matches the empty string at the beginning of a line</td></tr>
<tr><td><code>$</code></td><td><code>$</code></td><td>Matches the empty string at the end of a line</td></tr>
<tr><td><code>\<</code></td><td><code>\<</code></td><td>Matches the empty string at the beginning of a word</td></tr>
<tr><td><code>\></code></td><td><code>\></code></td><td>Matches the empty string at the end of a word</td></tr>
<tr><td><code>[</code>chars<code>]</code></td><td><code>[</code>chars<code>]</code></td><td>Match any character in the given class; if the first character after <code>[</code> is <code>^</code>, match any character not in the given class; a range of characters may be specified by first–last; for example, <code>\W</code> (below) is equivalent to the class <code>[^A-Za-z0-9]</code></td></tr>
<tr><td><code>\(\)</code></td><td><code>( )</code></td><td>Parentheses are used to override operator precedence</td></tr>
<tr><td><code>\</code>digit</td><td><code>\</code>digit</td><td><code>\n</code> matches a repeat of the text matched earlier in the regexp by the subexpression inside the nth opening parenthesis</td></tr>
<tr><td><code>\</code></td><td><code>\</code></td><td>Any special character may be preceded by a backslash to match it literally</td></tr>
<tr><td colspan="3">The following are for compatibility with GNU Emacs</td></tr>
<tr><td><code>\b</code></td><td><code>\b</code></td><td>Matches the empty string at the edge of a word</td></tr>
<tr><td><code>\B</code></td><td><code>\B</code></td><td>Matches the empty string if not at the edge of a word</td></tr>
<tr><td><code>\w</code></td><td><code>\w</code></td><td>Matches word-constituent characters (letters & digits)</td></tr>
<tr><td><code>\W</code></td><td><code>\W</code></td><td>Matches characters that are not word-constituent</td></tr>
</table>

Operator precedence is (highest to lowest) `?`, `*`, and `+`, concatenation, and finally `|`. All other constructs are syntactically identical to normal characters. For the truly interested, the file `dfa.c` describes (and implements) the exact grammar understood by the parser.

OPTIONS

-A *num*	Print *num* lines of context after every matching line
-B *num*	Print *num* lines of context before every matching line
-C	Print 2 lines of context on each side of every match
-*num*	Print *num* lines of context on each side of every match
-V	Print the version number on the diagnostic output
-b	Print every match preceded by its byte offset
-c	Print a total count of matching lines only
-e *expr*	Search for *expr*; useful if *expr* begins with –
-f *file*	Search for the expression contained in *file*
-h	Don't display filenames on matches
-i	Ignore case difference when comparing strings
-l	List files containing matches only
-n	Print each match preceded by its line number
-s	Run silently producing no output except error messages
-v	Print only lines that contain no matches for the `<expr>`
-w	Print only lines where the match is a complete word
-x	Print only lines where the match is a whole line

INCOMPATIBILITIES

The following incompatibilities with UNIX `grep` exist:

The context-dependent meaning of * is not quite the same (`grep` only).

-b prints a byte offset instead of a block offset.

The {*m*,*n*} construct of System V `grep` is not implemented.

AUTHORS

Mike Haertel wrote the deterministic `regexp` code and the bulk of the program.

James A. Woods is responsible for the hybridized search strategy of using Boyer-Moore-Gosper fixed-string search as a filter before calling the general regexp matcher.

Arthur David Olson contributed code that finds fixed strings for the aforementioned BMG search for a large class of regexps.

Richard Stallman wrote the backtracking regexp matcher that is used for *digit* backreferences, as well as the GNU `getopt`. The backtracking matcher was originally written for GNU Emacs.

D. A. Gwyn wrote the C `alloca` emulation that is provided so System V machines can run this program. (`alloca` is used only by RMS' backtracking matcher, and then only rarely, so there is no loss if your machine doesn't have a "real" `alloca`.)

Scott Anderson and Henry Spencer designed the regression tests used in the `regress` script.

Paul Placeway wrote the original version of this manual page.

COPYRIGHT

SEE ALSO

Utility Programs -	`emacs, ed, sh`
Documents -	*GNU Emacs Manual*

NAME

`groups` – show current group membership

SYNOPSIS

`groups`

DESCRIPTION

`groups` prints the names of each group in the current group list, along with the current effective group if that is not in the list. If a group ID is not recorded in the `/etc/group` file the number is printed.

SEE ALSO

System Calls - `getegid()`, `getgroups()`, `setgroups()`

NAME

`gzexe` – compress executable files in place

SYNOPSIS

gzexe [-d] [*name ...* **]**

DESCRIPTION

The `gzexe` utility allows you to compress executables in place and have them automatically uncompress and execute when you run them (at a penalty in performance). For example, if you execute `gzexe /bin/cat` it will create the following two files:

```
-r-xr-xr-x  1 root  bin   9644 Feb 11 11:16 /bin/cat
-r-xr-xr-x  1 bin   bin  24576 Nov 23 13:21 /bin/cat~
```

`/bin/cat~` is the original file and `/bin/cat` is the self-uncompressing executable file. You can remove `/bin/cat~` once you are sure that `/bin/cat` works properly. This utility is most useful on systems with very small disks.

OPTIONS

-d Decompress the given executables instead of compressing them.

CAVEATS

The compressed executable is a shell script. This may create some security holes. In particular, the compressed executable relies on the **PATH** environment variable to find `gzip` and some other utilities (`tail`, `chmod`, `ln`, `sleep`).

BUGS

`gzexe` attempts to retain the original file attributes on the compressed executable, but you may have to fix them manually in some cases, using `chmod` or `chown`.

SEE ALSO

Utility Programs - `gzip, znew, zmore, zcmp, zforce`

NAME

`gzip`, `gunzip`, `gzcat` – compress or expand files

SYNOPSIS

gzip [-acdfhlLnNrtvV19] [-S *suffix***] [***name* **...]**

gunzip [-acfhlLnLrtvV] [-S *suffix***] [***name* **...]**

gzcat [-fhLV] [*name* **...]**

DESCRIPTION

`gzip` reduces the size of the named files using Lempel-Ziv coding (LZ77). Whenever possible, each file is replaced by one with the extension `.gz`, while keeping the same ownership modes, access and modification times. If no files are specified, or if a file name is "-", the standard input is compressed to the standard output . `gzip` will only attempt to compress regular files. In particular, it will ignore symbolic links.

If the compressed file name is too long for its file system, `gzip` truncates it. `gzip` attempts to truncate only the parts of the file name longer than three characters. (A part is delimited by dots.) If the name consists of small parts only, the longest parts are truncated. For example, if file names are limited to 14 characters, `gzip.msdos.exe` is compressed to `gzi.msd.exe.gz`. Names are not truncated on systems which do not have a limit on file name length.

By default, `gzip` keeps the original file name and timestamp in the compressed file. These are used when decompressing the file with the **-N** option. This is useful when the compressed file name was truncated or when the time stamp was not preserved after a file transfer.

Compressed files can be restored to their original form using `gunzip` or `gzcat`. If the original name saved in the compressed file is not suitable for its file system, a new name is constructed from the original one to make it legal.

`gunzip` takes a list of files on its command line and replaces each file whose name ends with `.gz`, `-gz`, `.z`, `-z`, `_z` or `.Z` and which begins with the correct magic number with an uncompressed file without the original extension. gunzip also recognizes the special extensions .tgz and .taz as shorthands for .tar.gz and .tar.Z respectively. When compressing, gzip uses the .tgz extension if necessary instead of truncating a file with a .tar extension.

gunzip can currently decompress files created by gzip, zip, compress, compress **-H** or pack. The detection of the input format is automatic. When using the first two formats, gunzip checks a 32-bit CRC. For pack, gunzip checks the uncompressed length. The standard compress format was not designed to allow consistency checks. However gunzip is sometimes able to detect a bad .Z file. If you get an error when uncompressing a .Z file, do not assume that the .Z file is correct simply because the standard uncompress does not complain. This generally means that the standard uncompress does not check its input, and happily generates garbage output. The SCO compress **-H** format (LZH compression method) does not include a CRC but also allows some consistency checks.

Files created by zip can be uncompressed by gzip only if they have a single member compressed with the “deflation” method. This feature is only intended to help conversion of tar.zip files to the tar.gz format. To extract zip files with several members, use unzip instead of gunzip.

gzcat uncompresses either a list of files on the command line or its standard input and writes the uncompressed data on standard output. gzcat will uncompress files that have the correct magic number whether they have a .gz suffix or not.

gzip uses the Lempel-Ziv algorithm used in zip and PKZIP. The amount of compression obtained depends on the size of the input and the distribution of common substrings. Typically, text such as source code or English is reduced by 60-70%. Compression is generally much better than that achieved by LZW (as used in compress), Huffman coding (as used in pack), or adaptive Huffman coding (compact).

Compression is always performed, even if the compressed file is slightly larger than the original. The worst case expansion is a few bytes for the gzip file header, plus 5 bytes every 32K block, or an expansion ratio of 0.015% for large files. Note that the actual number of used disk blocks almost never increases. gzip preserves the mode, ownership and timestamps of files when compressing or decompressing.

OPTIONS

-a --ascii ASCII text mode: convert end-of-lines using local conventions. This option is supported only on some non-Unix systems.

-c --stdout --to-stdout
Write output on standard output; keep original files unchanged. If there are several input files, the output consists of a sequence of independently compressed members. To obtain better compression, concatenate all input files before compressing them.

-d --decompress --uncompress
Decompress.

-f --force Force compression or decompression even if the file has multiple links or the corresponding file already exists, or if the compressed data is read from or written to a terminal. If the input data is not in a format recognized by `gzip`, and if the option **--stdout** is also given, copy the input data without change to the standard output: let `gzcat` behave as `cat`. If **-f** is not given, and when not running in the background, `gzip` prompts to verify whether an existing file should be overwritten.

-h --help Display a help screen and quit.

-l --list For each compressed file, list the following fields:

compressed size
size of the compressed file

uncompressed size
size of the uncompressed file

ratio compression ratio (0.0% if unknown

uncompressed_name
name of the uncompressed file

The uncompressed size is given as -1 for files not in `gzip` format, such as compressed `.Z` files. To get the uncompressed size for such a file, you can use:

```
gzcat file.Z | wc -c
```

In combination with the **--verbose** option, the following fields are also displayed:

method	compression method
crc	the 32-bit CRC of the uncompressed data
date & time	time stamp for the uncompressed file

The compression methods currently supported are deflate, compress, `lzh` (SCO `compress` **-H**) and pack. The CRC is given as `0xffffffff` for a file not in `gzip` format.

With **--name**, the uncompressed name, date and time are those stored within the compress file if present.

With **--verbose**, the size total and compression ratio for all files is also displayed, unless some sizes are unknown.

With **--quiet**, the title and totals lines are not displayed.

-L --license Display the `gzip` license and quit.

-n --no-name When compressing, do not save the original file name and time stamp by default. (The original name is always saved if the name had to be truncated.) When decompressing, do not restore the original file name if present (remove only the `gzip` suffix from the compressed file name) and do not restore the original time stamp if present (copy it from the compressed file). This option is the default when decompressing.

-N --name When compressing, always save the original file name and time stamp; this is the default. When decompressing, restore the original file name and time stamp if present. This option is useful on systems which have a limit on file name length or when the time stamp has been lost after a file transfer.

-q --quiet Suppress all warnings.

-r --recursive Travel the directory structure recursively. If any of the file names specified on the command line are directories, `gzip` will descend into the directory and compress all the files it finds there (or decompress them in the case of `gunzip`).

-S .suf --suffix .suf
Use suffix .suf instead of `.gz`. Any suffix can be given, but suffixes other than `.z` and `.gz` should be avoided to avoid confusion when files are transferred to other systems. A null suffix forces `gunzip` to try decompression on all given files regardless of suffix, as in: `gunzip -S "" *`

Previous versions of `gzip` used the `.z` suffix. This was changed to avoid a conflict with `pack`.

-t --test Test. Check the compressed file integrity.

-v --verbose Verbose. Display the name and percentage reduction for each file compressed or decompressed.

-V --version Version. Display the version number and compilation options then quit.

-# --fast --best
Regulate the speed of compression using the specified digit #, where **-1** or **--fast** indicates the fastest compression method (less compression) and **-9** or **--best** indicates the slowest compression method (best compression). The default compression level is **-6** (that is, biased towards high compression at expense of speed).

ADVANCED USAGE

Multiple compressed files can be concatenated. In this case, `gunzip` will extract all members at once. For example:

```
gzip -c file1 > foo.gz
gzip -c file2 >> foo.gz
```

Then

```
gzcat foo
```

is equivalent to

```
cat file1 file2
```

In case of damage to one member of a `.gz` file, other members can still be recovered (if the damaged member is removed). However, you can get better compression by compressing all members at once:

```
cat file1 file2 | gzip > foo.gz
```

compresses better than

```
gzip -c file1 file2 > foo.gz
```

If you want to recompress concatenated files to get better compression:

```
gzcat old.gz | gzip > new.gz
```

If a compressed file consists of several members, the uncompressed size and CRC reported by the **`--list`** option applies to the last member only. If you need the uncompressed size for all members, you can use:

```
gzcat file.gz | wc -c
```

If you wish to create a single archive file with multiple members so that members can later be extracted independently, use an archiver such as `tar` or `zip`. GNU `tar` supports the **`-z`** option to invoke `gzip` transparently. `gzip` is designed as a complement to `tar`, not as a replacement.

ENVIRONMENT

The environment variable `GZIP` can hold a set of default options for `gzip`. These options are interpreted first and can be overwritten by explicit command line parameters. For example:

for `sh`:

```
GZIP="-8v --name"; export GZIP
```

for `csh`:

```
setenv GZIP "-8v --name"
```

DIAGNOSTICS

Exit status is normally `0`; if an error occurs, exit status is `1`. If a warning occurs, exit status is `2`.

`Usage: gzip [-cdfhlLnNrtvV19] [-S suffix] [file ...]`
: Invalid options were specified on the command line.

`file: not in gzip format`
: The file specified to gunzip has not been compressed.

`file: Corrupt input. Use gzcat to recover some data.`
: The compressed file has been damaged. The data up to the point of failure can be recovered using `gzcat` file > recover

`file: compressed with xx bits, can only handle yy bits`
: File was compressed (using LZW) by a program that could deal with more bits than the decompress code on this machine. Recompress the file with `gzip`, which compresses better and uses less memory.

`file: already has .gz suffix -- no change`
: The file is assumed to be already compressed. Rename the file and try again.

`file already exists; do you wish to overwrite (y or n)?`
: Respond `y` if you want the output file to be replaced; `n` if not.

`gzip: corrupt input`
: A `SIGSEGV` violation was detected which usually means that the input file has been corrupted.

`xx.x%`
: Percentage of the input saved by compression. (Relevant only for **-v** and **-l**.)

`-- not a regular file or directory: ignored`
: When the input file is not a regular file or directory, (e.g., a

symbolic link, socket, FIFO, device file), it is left unaltered.

`-- has xx other links: unchanged`

The input file has links; it is left unchanged. See `ln` for more information. Use the **-f** flag to force compression of multiply-linked files.

COPYRIGHT

The following copyright notice applies to the `gzip` program: Copyright 1992-1993 Jean-loop Gailly.

CAVEATS

When writing compressed data to a tape, it is generally necessary to pad the output with zeroes up to a block boundary. When the data is read and the whole block is passed to `gunzip` for decompression, `gunzip` detects that there is extra trailing garbage after the compressed data and emits a warning by default. You have to use the **--quiet** option to suppress the warning. This option can be set in the `GZIP` environment variable as in:

for `sh`:

```
GZIP="-q"  tar -xfz --block-compress /dev/rst0
```

for `csh`:

```
(setenv GZIP -q; tar -xfz --block-compr /dev/rst0
```

In the above example, `gzip` is invoked implicitly by the **-z** option of GNU `tar`. Make sure that the same block size (**-b** option of `tar`) is used for reading and writing compressed data on tapes. (This example assumes you are using the GNU version of `tar`.)

BUGS

The **--list** option reports incorrect sizes if they exceed 2GB. The **--list** option reports sizes as **-1** and CRC as `0xffffffff` if the compressed file is on a non seekable media.

In some rare cases, the **--best** option gives worse compression than the default compression level (**-6**). On some highly redundant files, `compress` compresses better than `gzip`.

SEE ALSO

Utility Programs - `znew, zcmp, zmore, zforce, gzexe`

NAME

`head` – print first few lines of a file

SYNOPSIS

head [– n] [*file ...* **]**

DESCRIPTION

`head` reads the named *files*, or standard input if none are given, and prints the first *n* lines on standard output. If *n* is not given, `head` prints the first ten lines.

SEE ALSO

Utility Programs - `less, tail`

NAME

`hfilt` – filter for HP printers

SYNOPSIS

`hfilt`

DESCRIPTION

The `hfilt` utility is a printer filter designed to be used with `lpd` so that text files may be correctly printed on HP printers, such as Laserjets and Inkjets. This filter is intended to be used with the `of` field of `/etc/printcap`.

The reason that text files need to be filtered for HP printers is that, in UNIX files, each line of output is typically delimited by only a newline. However, on HP printers, a newline will only move the print position to the next line. It will not move the print position to the beginning of that line. Lines of output on an HP printer need to be delimited by both a carriage return and a line feed.

In addition, the last page of output needs to be followed by a form feed to eject the page.

SEE ALSO

Utility Programs - `lpd`

File Formats - `printcap`

NAME

`hostname` – print or set name of current host system

SYNOPSIS

`hostname [` *name* **`]`**

DESCRIPTION

The `hostname` command prints the name of the current host. The super-user can set the `hostname` by giving an argument name. If *name* has been changed with `hostname`, then the new name will be seen at the next login or when the next shell is spawned.

SEE ALSO

System Calls - `gethostname()`

NAME

`id` – print user and group IDs and names

SYNOPSIS

`id`

DESCRIPTION

`id` prints the user ID, the user name, the group ID, and the group name of the parent process. If the real and effective IDs are different, then both are printed.

SEE ALSO

System Calls - `getgid()`, `getuid()`

NAME

`ident` – identify files

SYNOPSIS

`ident` *file ...*

DESCRIPTION

`ident` searches the named files for all occurrences of the pattern `$keyword:...$`, where `keyword` is one of

```
Author
Date
Header
Locker
Log
Revision
Source
State
```

These patterns are normally inserted automatically by the `RCS` command `co`, but can also be inserted manually.

`ident` works on text files as well as object files. For example, if the C program in file `f.c` contains

```
char rcsid[] = "$<Header-keyword>: Header information $";
```

and `f.c` is compiled into `f.o`, then the command

```
ident f.c f.o
```

will print

```
fc:
  $<Header-keyword>: Header information $
fo:
  $<Header-keyword>: Header information $
```

COPYRIGHT NOTICE

The RCS system is the work of Walter F. Tichy of Purdue University. It was obtained by Lynx from the Free Software Foundation. As such, Lynx Real-Time Systems will make available for nominal postage and materials cost complete source code for all parts of the RCS system.

Furthermore, LynxOS users are hereby permitted to copy and distribute RCS at will under these same terms; that is, notice that source is available from Lynx Real-Time Systems must be given along with any RCS distribution, or else the source itself.

If any RCS source is to be used as a component of some other software system, the Free Software Foundation should be contacted for details:

Free Software Foundation, Inc.
675 Massachusetts Ave.
Cambridge, MA 02139

Note that RCS as distributed by FSF has been modified in small ways by Lynx Real-Time Systems, Inc. While the source for RCS may be available elsewhere, it will be different from LynxOS RCS. Lynx Real-Time Systems assumes no responsibility for copies of RCS obtained elsewhere.

SEE ALSO

Utility Programs - `co, ident, rcs, rcsdiff, rcsintro, rlog`

File Formats - `rcsfile`

NAME

`ifconfig` – configure network interface parameters

SYNOPSIS

ifconfig *interface* *address_family* **[***address***[***dest_address***]]** **[***parameters***]**

ifconfig *interface* **[***protocol_family***]**

DESCRIPTION

`ifconfig` is used to assign an address to a network interface and/or configure network interface parameters. `ifconfig` must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters.

OPTIONS

address — For the DARPA-Internet family, the address is either a host name present in the host name data base, `hosts`(5), or a DARPA Internet address expressed in the Internet standard "dot notation".

address_family — Specifies the address family which affects interpretation of the remaining parameters. Since an interface can receive transmissions in differing protocols with different naming schemes, specifying the address family is recommeded. The address or protocol families currently supported is `inet`.

interface — Specify a string of the form *name-unit*, for example, `wd0`.

parameters — Specify one or more values from the following list as appropriate:

`alias` — Establish an additional network address for this interface. This is sometimes useful when changing network numbers, and one wishes to accept packets addressed to the old interface.

`arp` Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.

`-arp` Disable the use of the Address Resolution Protocol.

`broadcast` (Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

`debug` Enable driver dependent debugging code; usually, this turns on extra console error logging.

`-debug` Disable driver dependent debugging code.

`delete` Remove the network address specified. This would be used if you incorrectly specified an alias, or it was no longer needed. If you have incorrectly set an NS address having the side effect of specifying the host portion, removing all NS addresses will allow you to respecify the host portion.

`dest_address`
Specify the address of the correspondent on the other end of a point to point link.

`down` Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

`ipdst` This is used to specify an Internet host who is willing to receive IP packets encapsulating NS packets bound for a remote network. An apparent point to point link is constructed, and the address specified will be taken as the NS address and network of the destination. IP encapsulation of CLNP packets is done differently.

This option is not supported.

`metric` *n* Set the routing metric of the interface to n, default 0. The routing metric is used by the routing protocol (`routed`(1)). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

`netmask` *mask*
(Inet and ISO) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table `networks`(5). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

`nsellength` *n*
(ISO only) This specifies a trailing number of bytes for a received NSAP used for local identification, the remaining leading part of which is taken to be the NET (Network Entity Title). The default value is 1, which is conformant to US GOSIP. When an ISO address is set in an `ifconfig` command, it is really the NSAP which is being specified. For example, in US GOSIP, 20 hex digits should be specified in the ISO NSAP to be assigned to the interface. There is some

evidence that a number different from 1 may be useful for AFI 37 type addresses.

This option is not supported.

`trailers` Request the use of a "trailer" link level encapsulation when sending (default). If a network interface supports `trailers`, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.

This option is not supported.

`-trailers`
Disable the use of a "trailer" link level encapsulation.

This option is not supported.

`link[0-2]`
Enable special processing of the link level of the interface. These three options are interface specific in actual effect, however, they are in general used to select special modes of operation. An example of this is to enable SLIP compression. Currently, only used by SLIP.

`-link[0-2]`
Disable special processing at the link level with the specified interface.

`up` Mark an interface "up". This may be used to enable an interface after an "`ifconfig down`." It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

`ifconfig` displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, `ifconfig` will report only the details specific to that protocol family.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

Utility Programs - `netstat`, `routed`

HISTORY

The `ifconfig` command appeared in 4.2BSD.

NAME

`indent` – inserts/deletes whitespace in a C program

SYNOPSIS

indent [-bad] [-bap] [-bbb] [-bc] [-bl] [-bliN] [-br] [-cN] [-cdN] [-cdb] [-ce] [-ciN] [-cliN] [-cpN] [-cs] [-bs] [-dN] [-diN] [-fc1] [-fca] [-gnu] [-iN] [-ipN] [-kr] [-lN] [-lp] [-nbad] [-nbap] [-nbbb] [-nbc] [-ncdb] [-nce] [-ncs] [-nfc1] [-nfca] [-nip] [-nlp] [-npcs] [-npsl] [-nsc] [-nsob] [-nss] [-nv] [-orig] [-npro] [-pcs] [-psl] [-sc] [-sob] [-ss] [-st] [-T] [-tsN] [-v] [-*version***] [*file*]**

DESCRIPTION

This man page is based on the info file `indent.info`, produced by Makeinfo-1.47 from the input file `indent.texinfo`. This is Edition 0.02, 5 May 1992, of The `indent` Manual, for Indent Version 1.3.

COPYING

INTRODUCTION

The `indent` program can be used to make code easier to read. It can also convert from one style of writing C to another.

`indent` understands a substantial amount about the syntax of C, but it also attempts to cope with incomplete and misformed syntax.

In version 1.2 and more recent versions, the GNU style of indenting is the default.

INVOKING INDENT

As of version 1.3, the format of the `indent` command is:

indent [*options***] [***input-files***]**

indent [*options***] [***single-input-file***] [-o** *output-file***]**

This format is different from earlier versions and other versions of `indent`.

In the first form, one or more input files are specified. `indent` makes a backup copy of each file, and the original file is replaced with its indented version. `*Note Backup files::`, for an explanation of how backups are made.

In the second form, only one input file is specified. In this case, or when the standard input is used, you may specify an output file after the **-o** option.

To cause `indent` to write to standard output, use the **-st** option. This is only allowed when there is only one input file, or when the standard input is used.

If no input files are named, the standard input is read for input. Also, if the file name **-** is specified, then the standard input is read.

As an example, each of the following commands will input the program `slithy_toves.c` and write its indented text to `slithy_toves.out`:

```
indent slithy_toves.c -o slithy_toves.out
indent -st slithy_toves.c > slithy_toves.out
cat slithy_toves.c | indent -o slithy_toves.out
```

Most other options to `indent` control how programs are formatted. As of version 1.2, `indent` also recognizes a long name for each option name. Long options are prefixed by either **--** or `+`. `+` is being superseded by **--** to maintain consistency with the POSIX standard. In most of this document, the traditional, short names are used for the sake of brevity.

Here is another example:

```
indent -br test/metabolism.c -l85
```

This will indent the program test/metabolism.c using the **-br** and **-l85** options, write the output back to test/metabolism.c, and write the original contents of test/metabolism.c to a backup file in the directory test.

Equivalent invocations using long option names for this example would be:

```
indent --braces-on-if-line --line-length185 \
test/metabolism.c
```

```
indent +braces-on-if-line +line-length185 \
test/metabolism.c
```

If you find that you often use indent with the same options, you may put those options into a file called .indent.pro. indent will first look for .indent.pro in the current directory and use that if found. Otherwise, indent will search your home directory for .indent.pro and use that file if it is found. This behavior is different from that of other versions of indent, which load both files if they both exist.

Command line switches are handled *after* processing .indent.pro. Options specified later override arguments specified earlier, with one exception: Explicitly specified options always override background options (*note Common styles::.). You can prevent indent from reading an .indent.pro file by specifying the **-npro** option.

BACKUP FILES

As of version 1.3, GNU indent makes GNU--style backup files, the same way GNU Emacs does. This means that either simple or numbered backup file names may be made.

Simple backup file names are generated by appending a suffix to the original file name. The default for the this suffix is the one-character string ~ (tilde). Thus, the backup file for python.c would be python.c~.

Instead of the default, you may specify any string as a suffix by setting the environment variable SIMPLE_BACKUP_SUFFIX to your preferred suffix.

Numbered backup versions of a file `momewraths` look like `momewraths.c.~23~`, where 23 is the version of this particular backup. When making a numbered backup of the file `src/momewrath.c`, the backup file will be named `src/momewrath.c.~V~`, where V is one greater than the highest version currently existing in the directory `src`.

The type of backup file made is controlled by the value of the environment variable `VERSION_CONTROL`. If it is the string `simple`, then only simple backups will be made. If its value is the string `numbered`, then numbered backups will be made. If its value is numbered-existing, then numbered backups will be made if there `*already exist*` numbered backups for the file being indented; otherwise, a simple backup is made. If `VERSION_CONTROL` is not set, then `indent` assumes the behavior of numbered-existing.

Other versions of `indent` use the suffix `.BAK` in naming backup files. This behavior can be emulated by setting `SIMPLE_BACKUP_SUFFIX` to `.BAK`.

Note also that other versions of `indent` make backups in the current directory, rather than in the directory of the source file as GNU `indent` now does.

COMMON STYLES

There are several common styles of C code, including the GNU style, the Kernighan & Ritchie style, and the original Berkeley style. A style may be selected with a single background option, which specifies a set of values for all other options. However, explicitly specified options always override options implied by a background option.

As of version 1.2, the default style of GNU `indent` is the GNU style. Thus, it is no longer necessary to specify the option **`-gnu`** to obtain this format, although doing so will not cause an error. Option settings which correspond to the GNU style are:

```
-nbad -bap -nbbb -nbc -bl -bli2 -c33 -cd33 -ncdb -nce -cli0
-cp1 -di0 -nfc1 -nfca -i2 -ip5 -lp -pcs -psl -nsc -nsob
-nss -ts8
```

The GNU coding style is that preferred by the GNU project. It is the style that the GNU Emacs C mode encourages and which is used in the C portions of GNU Emacs. (People interested in writing programs for Project

GNU should get a copy of "The GNU Coding Standards", which also covers semantic and portability issues such as memory usage, the size of integers, etc.)

The Kernighan & Ritchie style is used throughout their well-known book "The C Programming Language". It is enabled with the **-kr** option. The Kernighan & Ritchie style corresponds to the following set of options:

```
-nbad -bap -nbbb -nbc -br -c33 -cd33 -ncdb -ce -ci4 -cli0
-cp33 -d0 -di1 -nfc1 -nfca -i4 -ip0 -l75 -lp -npcs -npsl
-nsc -nsob -nss -ts8
```

Kernighan & Ritchie style does not put comments to the right of code in the same column at all times (nor does it use only one space to the right of the code), so for this style `indent` has arbitrarily chosen column 33.

The style of the original Berkeley `indent` may be obtained by specifying **-orig** (or by specifying **--original**, using the long option name). This style is equivalent to the following settings:

```
-nbap -nbad -nbbb -bc -br -c33 -cd33 -cdb -ce -ci4 -cli0
-cp33 -d4 -di16 -fc1 -fca -i4 -ip4 -l75 -lp -npcs -psl -sc -nsob
-nss -ts8
```

BLANK LINES

Various programming styles use blank lines in different places. `indent` has a number of options to insert or delete blank lines in specific places.

The **-bad** option causes `indent` to force a blank line after every block of declarations. The **-nbad** option causes `indent` not to force such blank lines.

The **-bap** option forces a blank line after every procedure body. The **-nbap** option forces no such blank line.

The **-bbb** option forces a blank line before every block comment. A block comment is one which starts in column one when formatting of such comments is disabled, or one with - or * immediately following the `/*`. The **-nbbb** option does not force such blank lines.

The **-sob** option causes `indent` to swallow optional blank lines (that is, any optional blank lines present in the input will be removed from the output). If the **-nsob** is specified, any blank lines present in the input file will be copied to the output file.

The **-bad** option forces a blank line after every block of declarations. The **-nbad** option does not add any such blank lines.

For example, given the input:

```
char *foo;
char *bar;
/* This separates blocks of declarations.  */
int baz;
```

`indent` **-bad** produces:

```
char *foo;
char *bar;
/* This separates blocks of declarations.  */
int baz;
```

and `indent` **-nbad** produces:

```
char *foo;
char *bar;
/* This separates blocks of declarations.  */
int baz;
```

The **-bap** option forces a blank line after every procedure body. For example, given the input:

```
int
foo ()
{
  puts("Hi");
}
/* The procedure bar is even less interesting.  */
char *
bar ()
```

```
{
  puts("Hello");
}
```

`indent` **-bap** produces:

```
int
foo ()
{
  puts ("Hi");
}
/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts ("Hello");
}
```

and `indent` **-nbap** produces:

```
int
foo ()
{
  puts ("Hi");
}
/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts ("Hello");
}
```

No blank line will be added after the procedure `foo`.

COMMENTS

Comments are no longer formatted by default as of version 1.2. This can be enabled with the **-fca** option. Doing so will cause newlines in the comment text to be ignored and the line will be filled up to the length of a line (which can be modified with **-l**). When formatting is enabled, blank lines indicate paragraph breaks.

The **-fc1** option enables the formatting of comments which begin in the first column. The **-nfc1** option disables the formatting of first column comments. When comment formatting is disabled, overall comment indentation may still be adjusted.

The indentation of comments which do not appear to the right of code is set by the **-d** option, which specifies the number of spaces to the left of the surrounding code that the comment appears. For example, **-d2** places comments two spaces to the left of code; **-d0** lines up comments with the code. The **-cdb** option controls whether the `/*` and `*/` are placed on blank lines. With **-cdb**, comments look like this:

```
/*
 * this is a comment
 */
```

With **-ncdb**, comments look like this:

```
/* this is a comment */
```

-cdb only affects block comments, not comments to the right of code. The default is **-ncdb**.

Comments which appear on the same line as code are placed to the right. The column in which comments on code start is controlled by the **-c** option. The column in which comments to the right of declarations start is controlled by the **-cd** option. By default, they start in the same column as comments to the right of code, which is column 33. The column number for comments to the right of `#else` and `#endif` statements is controlled by the **-cp** option. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

If the **-sc** option is specified, `*` is placed at the left edge of all comments. For example:

```
/* This is a comment which extends from one line
 * onto the next line, thus causing us to consider
 * how it should continue.  */
```

instead of

```
/* This is a comment which extends from one line
   onto the next line, thus causing us to consider
   how it should continue.  */
```

STATEMENTS

The **-br** or **-bl** option specifies how to format braces. The **-br** option formats braces like this:

```
if (x > 0) {
  x--;
}
```

The **-bl** option formats them like this:

```
if (x > 0)
  {
    x--;
  }
```

If you use the **-bl** option, you may also want to specify the **-bli** option. This option specifies the number of spaces by which braces are indented. **-bli2**, the default, gives the result shown above. **-bli0** results in the following:

```
if (x > 0)
{
  x--;
}
```

If you are using the **-br** option, you probably want to also use the **-ce** option. This causes the `else` in an if-then-else construct to cuddle up to the immediately preceding `}`. For example, with **-br -ce** you get the following:

```
if (x > 0) {
  x--;
} else {
  fprintf (stderr, "...something wrong?\n");
}
```

With **-br -nce** that code would appear as

```
if (x > 0) {
  x--;
}
else {
  fprintf (stderr, "...something wrong?\n");
}
```

The **-cli** option specifies the number of spaces that case labels should be indented to the right of the containing `switch` statement.

If a semicolon is on the same line as a `for` or `while` statement, the **-ss** option will cause a space to be placed before the semicolon. This emphasizes the semicolon, making it clear that the body of the `for` or `while` statement is an empty statement. **-nss** disables this feature.

The **-pcs** option causes a space to be placed between the name of the procedure being called and the `(` (for example, `puts ("Hi");`. The **-npcs** option would give `puts("Hi");`).

If the **-cs** option is specified, `indent` puts a space after a cast operator.

The **-bs** option ensures that there is a space between the keyword `sizeof` and its argument. In some versions, this is known as the `Bill_Shannon` option.

DECLARATIONS

By default `indent` will line up identifiers, in the column specified by the **-di** option. For example, **-di16** makes things look like:

```
int foo;
char* bar;
```

Using a small value (such as one or two) for the **-di** option can be used to cause the indentifiers to be placed in the first available position, for example

```
int foo;
char* bar;
```

The value given to the **-di** option will still affect variables which are put on separate lines from their types, for example **-di2** will lead to

```
int
  foo;
```

If the **-bc** option is specified, a newline is forced after each comma in a declaration. For example,

```
int a,
  b,
  c;
```

With the **-nbc** option this would look like

```
int a, b, c;
```

The **-psl** option causes the type of a procedure being defined to be placed on the line before the name of the procedure. This style is required for the `etags` program to work correctly, as well as some of the `c-mode` functions of Emacs.

If you are not using the **-di1** option to place variables being declared immediately after their type, you need to use the **-T** option to tell `indent` the name of all the type names in your program that are defined by `typedef`. **-T** can be specified more than once, and all names specified are used. For example, if your program contains

```
typedef unsigned long CODE_ADDR;
typedef enum {red, blue, green} COLOR;
```

you would use the options `-T CODE_ADDR -T COLOR`.

INDENTATION

One issue in the formatting of code is how far each line should be indented from the left margin. When the beginning of a statement such as `if` or `for` is encountered, the indentation level is increased by the value specified by the **-i** option. For example, use **-i8** to specify an eight character indentation for each level. When a statement is continued from a previous line, it is indented by a number of additional spaces specified by the **-ci** option. **-ci** defaults to 0. However, if the **-lp** option is specified, and a line has a left parenthesis which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left parenthesis. This processing also applies to `[` and applies to `{` when it occurs in initialization lists. For example, a piece of continued code might look like this with **-nlp -ci3** in effect:

```
p1 = first_procedure (second_procedure (p2, p3),
  third_procedure (p4, p5));
```

With **-lp** in effect the code looks somewhat clearer:

```
p1 = first_procedure (second_procedure (p2, p3),
  third_procedure (p4, p5));
```

`indent` assumes that tabs are placed at regular intervals of both input and output character streams. These intervals are by default 8 columns wide, but (as of version 1.2) may be changed by the **-ts** option. Tabs are treated as the equivalent number of spaces.

The indentation of type declarations in old-style function definitions is controlled by the **-ip** parameter. This is a numeric parameter specifying how many spaces to indent type declarations. For example, the default **-ip5** makes definitions look like this:

```
char *
create_world (x, y, scale)
  int x;
  int y;
  float scale;
{
  . . .
}
```

For compatibility with other versions of indent, the option **-nip** is provided, which is equivalent to **-ip0**.

MISCELLANEOUS OPTIONS

To find out what version of `indent` you have, use the command `indent -version`. This will report the version number of `indent`, without doing any of the normal processing.

The **-v** option can be used to turn on verbose mode. When in verbose mode, `indent` reports when it splits one line of input into two more lines of output, and gives some size statistics at completion.

OPTIONS

The options for `indent` are shown in the table "Indent Options Alphabetized By Short Options", alphabetized by short option. This table is followed by a cross key alphabetized by long option.

Table 1: Indent Options Alphabetized By Short Options

Indent Options (Short/Long)	Description
`-bad` `--blank-lines-after-declarations`	Force blank lines after the declarations
`-bap` `--blank-lines-after-procedures`	Force blank lines after procedure bodies
`-bbb` `--blank-lines-after-block-comments`	Force blank lines after block comments
`-bc` `--blank-lines-after-commas`	Force newline after comma in declaration
`-bl` `--braces-after-if-line`	Put braces on line after if, etc.
`-bliN` `--brace-indentN`	Indent braces N spaces
`-br` `--braces-on-if-line`	Put braces on line with if, etc.
`-cN` `--comment-indentationN`	Put comments to the right of code in column N
`-cdN` `--declaration-comment-columnN`	Put comments to the right of the declarations in column N
`-cdb` `--comment-delimiters-on-blank-lines`	Put comment delimiters on blank lines
`-ce` `--cuddle-else`	Cuddle else and preceding }

Table 1: Indent Options Alphabetized By Short Options

Indent Options (Short/Long)	Description
`-ciN` `--continuation-indentationN`	Continuation indent of N spaces
`-cliN` `--case-indentationN`	Case label indent of N spaces
`-cpN` `--else-endif-columnN`	Put comments to the right of #else and #endif statements in column N
`-cs` `--space-after-cast`	Put a space after a cast operator
`-bs` `--blank-before-sizeof`	Put a space between sizeof and its argument
`-dN` `--line-comments-indentationN`	Set indentation of comments not to the right of code to N spaces
`-diN` `--declaration-indentationN`	Put variables in column N
`-fc1` `--format-first-column-comments`	Format comments in the first column
`-fca` `--format-all-comments`	Do not disable all formatting of comments
`-gnu` `--gnu-style`	Use GNU coding style. This is the default
`-iN` `--indent-levelN`	Set indentation level to N spaces
`-ipN` `--parameter-indentationN`	Indent parameter types in old-style function definitions by N spaces
`-kr` `--k-and-r-style`	Use Kernighan & Ritchie coding style
`-lN` `--line-lengthN`	Set maximum line length to N

Table 1: Indent Options Alphabetized By Short Options

Indent Options (Short/Long)	Description
-lp --continue-at-parentheses	Line up continued lines at parentheses
-nbad --no-blank-lines-after-declarations	Do not force blank lines after declarations
-nbap --no-blank-lines-after-procedures	Do not force blank lines after procedure bodies
-nbbb --no-blank-lines-after-block-comments	Do not force blank-lines after block comments
-nbc --no-blank-lines-after-commas	Do not force newlines after commas in declarations
-ncdb --no-comment-delimiters-on-blank-lines	Do not put comment delimiters on blank lines
-nce --dont-cuddle-else	Do not cuddle } and else
-ncs --no-space-after-casts	Do not put a space after cast operators
-nfc1 --dont-format-first-column-comments	Do not format comments in the first column as normal
-nfca --dont-format-comments	Do not format any comments
-nip --no-parameter-indentation	Zero width indentation for parameters
-nlp --dont-line-up-parentheses	Do not line up parentheses
-npcs --no-space-after-function-call-names	Do not put space after the function in function calls
-npsl --dont-break-procedure-type	Put the type of a procedure on the same line as its name

Table 1: Indent Options Alphabetized By Short Options

Indent Options (Short/Long)	Description
`-nsc` `--dont-star-comments`	Do not put the * character at the left of comments
`-nsob` `--leave-optional-blank-lines`	Do not swallow optional blank lines
`-nss` `--dont-space-special-semicolon`	Do not force a space before the semicolon after certain statements. Disables `-ss`
`-nv` `--no-verbosity`	
`-orig` `--original`	Use the original Berkeley coding style
`-npro` `--ignore-profile`	Do not read .indent.pro files
`-pcs` `--space-after-procedure-calls`	Insert a space between the name of the procedure being called and the (
`-psl` `--procnames-start-lines`	Put the type of a procedure on the line before its name
`-sc` `--start-left-side-of-comments`	Put the * character at the left of comments
`-sob` `--swallow-optional-blank-lines`	Swallow optional blank lines
`-ss` `--space-special-semicolon`	On one-line for and while statements, force a blank before the semicolon
`-st` `--standard-output`	Write to standard output
`-T`	Tell indent the name of type names

Table 1: Indent Options Alphabetized By Short Options

Indent Options (Short/Long)	Description
-tsN --tab-sizeN	Set tab size to N spaces
-v --verbose	Enable verbose mode
-version	Output the version number of indent

OPTIONS CROSS-REFERENCE

Within the table "Indent Options Cross-Referenced From Long To Short"is a list of options alphabetized by long option, to help you find the corresponding short option.

Table 2: Indent Options Cross-Referenced From Long To Short

Long	Short
--blank-lines-after-block-comments	-bbb
--blank-lines-after-commas	-bc
--blank-lines-after-declarations	-bad
--blank-lines-after-procedures	-bap
--braces-after-if-line	-bl
--brace-indent	-bli
--braces-on-if-line	-br
--case-indentation	-cliN
--comment-delimiters-on-blank-lines	-cdb
--comment-indentation	-cN
--continuation-indentation	-ciN
--continue-at-parentheses	-lp
--cuddle-else	-ce
--declaration-comment-column	-cdN

Table 2: Indent Options Cross-Referenced From Long To Short

Long	Short
`--declaration-indentation`	`-diN`
`--dont-break-procedure-type`	`-npsl`
`--dont-cuddle-else`	`-nce`
`--dont-format-comments`	`-nfca`
`--dont-format-first-column-comments`	`-nfc1`
`--dont-line-up-parentheses`	`-nlp`
`--dont-space-special-semicolon`	`-nss`
`--dont-star-comments`	`-nsc`
`--else-endif-column`	`-cpN`
`--format-all-comments`	`-fca`
`--format-first-column-comments`	`-fc1`
`--gnu-style`	`-gnu`
`--ignore-profile`	`-npro`
`--indent-level`	`-iN`
`--k-and-r-style`	`-kr`
`--leave-optional-blank-lines`	`-nsob`
`--line-comments-indentation`	`-dN`
`--line-length`	`-lN`
`--no-blank-lines-after-block-comments`	`-nbbb`
`--no-blank-lines-after-commas`	`-nbc`
`--no-blank-lines-after-declarations`	`-nbad`
`--no-blank-lines-after-procedures`	`-nbap`
`--no-comment-delimiters-on-blank-lines`	`-ncdb`
`--no-space-after-casts`	`-ncs`
`--no-parameter-indentation`	`-nip`
`--no-space-after-function-call-names`	`-npcs`

Table 2: Indent Options Cross-Referenced From Long To Short

Long	Short
`--no-verbosity`	`-nv`
`--original`	`-orig`
`--parameter-indentation`	`-ipN`
`--procnames-start-lines`	`-psl`
`--space-after-cast`	`-cs`
`--space-after-procedure-calls`	`-pcs`
`--space-special-semicolon`	`-ss`
`--standard-output`	`-st`
`--start-left-side-of-comments`	`-sc`
`--swallow-optional-blank-lines`	`-sob`
`--tab-size`	`-tsN`
`--verbose`	`-v`

COPYRIGHT

The following copyright notice applies to the `indent` program.

RETURN VALUE

Unknown

FILES

`$HOME/.indent.pro` holds default options for indent.

AUTHOR

The Free Software Foundation.

HISTORY

Derived from the UCB program `indent`.

NAME

inetd – Internet super-server

SYNOPSIS

inetd [-d] [*config* **]**

DESCRIPTION

inetd should be run at boot time by rc.network. It then listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request. After the program is finished, it continues to listen on the socket (except in some cases which will be described below). Essentially, inetd allows running one daemon to invoke several others, reducing load on the system.

Upon execution, inetd reads its configuration information from a configuration file which, by default, is /etc/inetd.conf. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a "#" at the beginning of a line. There must be an entry for each field. The fields of the configuration file are as follows:

```
name type protocol wait/nowait user program args
```

The service name entry is the name of a valid service in the file /etc/services. For "internal" services (discussed below), the service name must be the official name of the service (that is, the first entry in /etc/services).

The socket type should be one of stream, dgram, raw, rdm, or seqpacket, depending on whether the socket is a stream, datagram, raw, reliably delivered message, or sequenced packet socket.

The protocol must be a valid protocol as given in /etc/protocols. Examples might be tcp or udp.

The wait/nowait entry is applicable to datagram sockets only (other sockets should have a nowait entry in this space). If a datagram server connects to its peer, freeing the socket so inetd can received further messages on the socket, it is said to be a "multi-threaded" server, and should use the `nowait` entry. For datagram servers which process all incoming datagrams on a socket and eventually time out, the server is said to be "single-threaded" and should use a `wait` entry. `comsat` (`biff`) and `talk` are both examples of the latter type of datagram server. `tftpd` is an exception; it is a datagram server that establishes pseudo-connections. It must be listed as `wait` in order to avoid a race; the server reads the first packet, creates a new socket, and then forks and exits to allow `inetd` to check for new service requests to spawn new servers.

The `user` entry should contain the user name of the user as whom the server should run. This allows for servers to be given less permission than root. The `server program` entry should contain the pathname of the program which is to be executed by `inetd` when a request is found on its socket. If `inetd` provides this service internally, this entry should be "internal".

The arguments to the server program should be just as they normally are, starting with `argv[0]`, which is the name of the program. If the service is provided internally, the word "internal" should take the place of this entry.

`inetd` provides several "trivial" services internally by use of routines within itself. These services are `echo`, `discard`, `chargen` (character generator), `daytime` (human readable time), and `time` (machine readable time, in the form of the number of seconds since midnight GMT, January 1, 1900). All of these services are TCP-based. For details of these services, consult the appropriate RFC from the Network Information Center.

`inetd` rereads its configuration file when it receives a hangup signal, `SIGHUP`. Services may be added, deleted or modified when the configuration file is reread.

SEE ALSO

Utility Programs -	`fingerd, ftpd, rexecd, rlogind, rshd, telnetd, tftpd`
File Formats -	`inetd.conf, services, protocols`

NAME

init - master system process timesharing

DESCRIPTION

init is started by the operating system as the initial master process. It refuses to be run otherwise. init is responsible for starting the system initialization program rc (a shell script), starting a single-user shell at boot time if necessary, and starting and monitoring multi-user operations.

When the operating system starts init, it passes it an integer containing several boot parameters, as described in reboot.h. init responds to these flags as follows:

RB_AUTOBOOT — init does not start an initial single-user mode shell, and includes the character "**a**" in the argument it passes to rc. The standard rc responds to the "**a**" by refraining from asking questions of the operator at boot time. If this flag is not set, init first starts a single user shell, then starts rc after that shell terminates. The "**a**" will not be passed to rc, which will therefore request various information and confirmations from the operator at the console.

RB_NOFSCK — The letter "**f**" is included in the argument passed to rc, causing it to perform file system checks with fsck on each file system named in /etc/fstab. Note that the name of this reboot flag, RB_NOFSCK, is completely backwards.

RB_MFSCK — This flag is set by the operating system if the LynxOS non-volatile disk cache contains invalid information at boot time. The character "**m**" is included in the argument to rc, which instructs it to perform file system checks regardless of the state of RB_NOFSCK.

RB_MAKENODES — If set, init starts the mknod program in the /dev directory with the argument **–a**, causing it to make all special files named in /etc/nodetab. init does this before proceeding with any other operations. This switch is generally used after installing a new operating system configuration with new devices.

As mentioned, the standard `rc` is a shell script. It can be easily modified or replaced in order to reflect local boot requirements.

When `init` starts up it reads the file `/etc/starttab` to obtain the name of the single user mode shell (usually `/bin/dlsh`), the name of the `rc` program (usually `/bin/rc`), and default values for core, stack, and data size limits for all processes.

When conducting multi-user operations, `init` uses information in the `/etc/ttys` file to determine which serial ports should be enabled for use, and what process to start for each enabled port.

`init` can be controlled to some extent by sending it signals. The `SIGHUP` signal causes `init` to re-read the `/etc/ttys` file. This signal should be sent if a terminal is enabled or disabled by changing the file. The `SIGQUIT` signal disables further logins, essentially disabling all serial ports. The `SIGINT` signal tells `init` to kill all child processes and then itself. It is unwise to send `init` the `SIGKILL` or `SIGSTOP` signals, as this will cause the operating system to panic.

SEE ALSO

Utility Programs -	`reboot`
System Calls -	`reboot(), sigvec()`
File Formats -	`starttab, ttys`

NAME

`initrarp` – initialize ARP table through RARP

SYNOPSIS

`initrarp [ -f` *altdir* **`]`**

DESCRIPTION

Use the `initrarp` command to initialize an operating system server's ARP table. The Address Resolution Protocol (`arp`) and the Reverse Address Resolution Protocol (`rarp`) use the ARP table to maintain Ethernet-to-Internet address translation information for all diskless clients. That information is used by the server when you boot a diskless client.

The `initrarp` command searches the `/tftpboot` directory for each diskless client's second-stage bootstrap file. The name of each of these files is a hexadecimal representation of a client's internet address. For every entry in `/tftpboot`, `initrarp` finds the hostname that corresponds to the Internet address and then extracts the corresponding Ethernet address of that host from the `/etc/ethers` file. This Ethernet-Internet address pair is then placed as a permanent entry in the server's ARP table.

With the **`-f`** flag, `initrarp` uses the directory named *altdir* instead of `/tftpboot` to scan for second-stage bootstrap files. Usually, this flag is not necessary. The superuser can invoke this command at any time she or he adds an entry for another diskless workstation.

NOTE

The Internet and Ethernet address of every diskless client must be present in the `/etc/hosts` and `/etc/ethers` files.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `arp`

File Formats - `hosts, ethers`

NAME

`install` – intelligent copy for software installation

SYNOPSIS

`install [-c] [-m` *mode***`] [-u`** *user***`] [-g`** *group***`]`** *source destination*

DESCRIPTION

`install` moves or copies the source file to the *destination* file after removing any existing *destination*. Unless otherwise requested, the ownership and permission information of source will be kept by the new *destination* file. If *destination* is a directory, then the final name is constructed by appending the base name of *source* to the directory name.

`install` is generally used in makefiles to copy a program from a development directory to the actual production directory. Because the current version may be in use, its executable file must be removed before the new version can be moved into place.

If *source* is "–", then standard input is read; *destination* must not be a directory name. This form is particularly useful when installing a program over a network interface via `rsh`:

```
rsh remote install -m 755 -u root - /bin/myprog < myprog
```

OPTIONS

`-c` The *source* file is copied to the *destination* rather than being renamed. Thus two copies of the file will exist.

`-g` *group* Group ownership for *destination* is set to the given *group* number or name.

`-m` *mode* The destination file is given the permissions described by the octal number *mode*, as per `chmod`.

`-u` *user* Ownership for *destination* is set to the given *user* number or name.

SEE ALSO

Utility Programs - `chmod, chown, chgrp, cp, ln, rm`

NAME

`ipcrm` – interprocess communication facilities removal

SYNOPSIS

ipcrm [[-q *msqid***] [-m** *shmid***] [-s** *semid***] [-Q** *msgkey***]**
[-M *shmkey***] [-S** *semkey***] ...]**

DESCRIPTION

`ipcrm` removes one or more System V-compatible interprocess communication (IPC) facility entries. System V-compatible IPC facilities include message queues, shared memory, and semaphores. The different options allow an entry to be removed by providing either its identifier or the key with which it was created:

-q *msqid* Removes the message queue with identifier *msqid*.

-m *shmid* Removes the shared memory segment with identifier *shmid*.

-s *semid* Removes the semaphore set with identifier *semid*.

-Q *msgkey* Removes the message queue that was created with key *msgkey*.

-M *shmkey* Removes the shared memory segment that was created with key *shmkey*.

-S *semkey* Removes the semaphore set that was created with key *semkey*.

`ipcrm` cannot use *msgkey*, *shmkey*, or *semkey* to remove a message queue, shared memory segment, or semaphore created with a key equal to 0.

The System V-compatible semaphore facilities are functionally similar to, but otherwise completely different from, the LynxOS semaphore facilities.

SEE ALSO

Utility Programs - `ipcs`

System Calls - `msgctl()`, `semctl()`, `shmctl()`

NAME

`ipcs` – interprocess communication facilities status

SYNOPSIS

`ipcs [-qms] [-bcopta]`

DESCRIPTION

`ipcs` shows the status of all active System V-compatible interprocess communication (IPC) facilities on the system. System V-compatible IPC facilities include message queues, shared memory, and semaphores. The first group of options (**`-qms`**) determines which of the three facility types to include in the report:

`-q` Show status of all active System V-compatible message queues.

`-m` Show status of all active System V-compatible shared memory segments.

`-s` Show status of all active System V-compatible semaphores sets.

These three options can be used one at a time, in combination, or not at all. Specifying none has the same effect as specifying all three.

The second group of options (**`-bcopta`**) determines what kind of information to show about each facility:

`-b` For message queues, show the maximum number of bytes allowed (`QBYTES`) on each message queue. For shared memory, show the size (`SEGSZ`) of each shared memory segment. For semaphores, show the number of semaphores (`NSEMS`) in each semaphore set.

`-c` Show the creator's user ID (`CREATOR`) and group (`CGROUP`).

`-o` For message queues, show the number of bytes (`CBYTES`) and number of messages (`QNUM`) currently on each message queue. For shared memory, show the number of processes attached (`NATTCH`) to each shared memory segment. For semaphores, no additional information is shown.

-p For message queues, show the process ID of the last process to send a message (`LSPID`) and the process ID of the last process to receive a message (`LRPID`) from each message queue. For shared memory, show the process ID of the creator (`CPID`) of each shared memory segment and the process ID of the last process to attach or detach (`LPID`) each shared memory segment. For semaphores, no additional information is shown.

-t For message queues, show the time the last message was sent (`STIME`) to each message queue, the time the last message was received (`RTIME`) from each message queue, and the time each message queue was created or changed (`CTIME`). For shared memory, show the time of the last attach (`ATIME`) to each shared memory segment, the time of the last detach (`DTIME`) from each shared memory segment, and the time each shared memory segment was created or changed (`CTIME`). For semaphores, show the time of the last semaphore operation (`OTIME`) on each semaphore set and the time each semaphore set was created or changed (`CTIME`).

-a Use all options from the second set. This is equivalent to **-bcopt**.

The output of `ipcs` is displayed in columns with headings at the top of each column. Certain columns are always displayed regardless of the options, while others are displayed only in response to certain options. The meaning of each column is as follows:

`T` The facility type:

- **q** System V-compatible message queue.
- **m** System V-compatible shared memory segment.
- **s** System V-compatible semaphore set.

`ID` The identifier of the facility. This is an integer.

KEY
: The key used as an argument to `msgget`, `semget`, or `shmget` to create or retrieve the facility. For a shared memory segment that has been removed while process are still attached to it, the key will be `IPC_PRIVATE`.

MODE
: The access modes and flags of the facility shown as an 11-character string. The first two characters are interpreted differently depending on the type of the facility. For message queues, the interpretation is as follows:

First character:

S a process is waiting on `msgsnd` to the message queue.

\- otherwise.

Second character:

R a process is waiting on `msgrcv` from the message queue.

\- otherwise.

For shared memory segments, the interpretation is as follows:

First character:

D the shared memory segment has been removed but processes are still attached to it.

\- otherwise.

Second character:

C the shared memory segment is to be zeroed the first time it is attached.

\- otherwise.

For semaphore sets, the first two characters are unused so each is shown as a dash "–". The last nine characters are interpreted as three sets of three bits each. The three sets represent, in order, the access permissions of the facility's owner, of users in the owner's group, and of all other users.

The first character in each set tells whether read permission is allowed. The meaning of the second character in each set depends on the facility type. For message queues and shared memory, it means write permission. For semaphores, it means alter permissions. The third character in each set is unused and always appears as a dash "**-**". The characters indicating these permissions are as follows:

r	read permission is allowed.
w	write permission is allowed (message queues and shared memory).
a	alter permission is allowed (semaphores).
-	permission is not allowed.

OWNER	The login name of the facility's owner.
GROUP	The login group of the facility's owner.
CREATOR	The login name of the facility's creator.
CGROUP	The login group of the facility's creator.
CBYTES	The number of bytes currently on the message queue.
QNUM	The number of messages currently on the message queue.
QBYTES	The maximum number of bytes allowed on the message queue.
LSPID	The process ID of the last process to send a message to the queue.
LRPID	The process ID of the last process to receive a message from the queue.
STIME	The time the last message was sent to the message queue.

`RTIME`	The time the last message was received from the message queue.
`CTIME`	The time the facility entry was created or changed.
`NATTCH`	The number of processes attached to the shared memory segment.
`SEGSZ`	The size of the shared memory segment.
`CPID`	The process ID of the process that created the shared memory segment.
`LPID`	The process ID of the last process to attach or detach the shared memory segment.
`ATIME`	The time of the last attach to the shared memory segment.
`DTIME`	The time of the last detach from the shared memory segment.
`NSEMS`	The number of semaphores in the semaphore set.
`OTIME`	The time of the last semaphore operation on the semaphore set.

The System V-compatible semaphore facilities are functionally similar to but otherwise completely different from the LynxOS semaphore facilities.

SEE ALSO

Utility Programs -	`ipcrm, sems, smems`
System Calls -	`msgctl(), msgget(), msgrcv(), msgsnd(), semctl(), semget(), semop(), shmat(), shmctl(), shmdt(), shmget()`

NAME

`join` – relational database operator

SYNOPSIS

join [-a *filenumber***] [-e** *string***] [-j** *filenumber field***] [-o** *list***]**
[-t *char***]** *filename1 filename2*

DESCRIPTION

The `join` utility performs an “equality join” on the files *filename1* and *filename2*. The joined fields are written to the standard output. The files *filename1* and *filename2* should be ordered in the collating sequence of `sort -b` on the fields on which they are to be joined (by default the first in each line). The default input field separators are <space>, <tab> and <newline>. Multiple field separators are counted as one and leading separators are ignored.

OPTIONS

-a *filenumber* In addition to the default output, produce a line for each unpairable line in the file *filenumber*.

-e *string* Replace empty output fields by *string*.

-j *filenumber field*
Join on the fieldth *field* of *filenumber*. If *filenumber* is missing, use the fieldth *field* in each file.

-o *list* Each output line comprises the fields specified in *list*, and each space separated element has the form *filenumber.field*. The *list* is written only for lines with matching `join` fields.

-t *char* Use the character *char* as a separator, both for input and output.

NAME

`kconfig` – produce a kernel makefile

SYNOPSIS

kconfig [-t *target* **] [-s** *suffix* **] [-S** *stub qualifier* **]** *file modules*

DESCRIPTION

`kconfig` reads the kernel configuration file *file* to determine which of the object files in *modules* should be replaced with stubs. It then creates a `makefile` and writes it to the standard output. Stdout should be redirected to a file called `kernel.makefile` if the supplied makefiles will be used. The normal kernel configuration file is `KERNEL.CFG`. *modules* is a master list of all object modules that are linked together to form the LynxOS kernel. Stubs make it possible to remove certain unwanted features from the LynxOS kernel.

OPTIONS

-t *target* — *target* is the name to be used for the executable file created by `ld`. The default is `a.out`.

-s *suffix* — *suffix* is the suffix used for the object files. The default is `.o`.

-S *stub qualifier* — *stub qualifier* is the portion of the stub file name that identifies it as a stub file. The default is `.stub`.

NOTE

The file `kernel.makefile` is automatically produced if the supplied makefiles are used to build the LynxOS kernel.

SEE ALSO

Utility Programs - `make`

File Formats - `kconfig`

NAME

`keyb` – international keyboard support

SYNOPSIS

`keyb [` *KeyboardLayoutFile* **`]`**

DESCRIPTION

The `keyb` utility is designed to adapt the LynxOS for x86 standard `atc` console driver to the various national variants of the PC/AT-style keyboard.

The *KeyboardLayoutFile* is a simple text file defining the characters or code values to be generated when a specific key is hit. The syntax is explained in `keyb` under File Formats. `keyb` reads the *KeyboardLayoutFile* and passes the data to `stdout` which is expected to be the `atc` console device driver. The driver will then use the data for encoding the keystrokes.

If *KeyboardLayoutFile* is not given as an argument, then `keyb` will reset the driver to its built-in defaults which correspond to the US standard layout of the PC/AT-style keyboard.

If *KeyboardLayoutFile* is not found then the default directory `/reusr/local/lib/keyb/` is checked for this file. If the *KeyboardLayoutFile* isn't there either, then `keyb` will abort with an error message.

NOTE

The `atc` console driver included in LynxOS releases older than Release 2.2 did not support internationalization. The `keyb` utility will not work unless the `atc` console driver supports internationalization.

FILE

`/usr/local/doc/atc-4X-chartab`

DIAGNOSTICS

The error messages returned from `keyb` are self-explanatory.

SEE ALSO

Device Facilities - `atc`

File Formats - `keyb`

NAME

`kill` – terminate a process

SYNOPSIS

kill [–*signal***]** *processID ...*

kill –l

kill –t *threadID ...*

kill [–*signal***] –g** *processID groupID ...*

DESCRIPTION

`kill` normally sends a `TERM` signal (signal 15) to the specified processes. If a signal name or number preceded by a "**–**"is given as the first argument, then `kill` will send that signal to the specified processes instead of the `TERM` signal. Any process ID can be preceded by a **–g** argument to force the ID to be taken as a process group number, or by a **–t** argument to force the ID to be taken as a thread ID.

The **–l** option causes `kill` to print a list of available signal names and numbers. The names listed can be given as arguments with or without the `SIG` prefix, and in upper or lower case.

The `TERM` signal might not kill processes that trap it. The un-trapable `KILL` signal should be sent if `TERM` is ineffective.

If process 0 is given as a target, then all members in the current process group receive the signal (except for the `kill` process itself). If process **–1** is given, then all processes in the system except process 1 (usually) and the `kill` process receive the signal.

Process numbers of background processes can be found by using the `ps` utility. The shell reports process group numbers of background jobs when the job is started. Thread IDs can be found by using the `ps` utility with the **–t** option.

FILES

`/dev/mem`	memory device, used to get a thread's process ID.

SEE ALSO

Utility Programs -	`dlsh, ps`
System Calls -	`kill(), sigvec(), wait()`
Library Functions -	`signal()`

NAME

`l-cc` – legacy Lynx C compiler (x86 and 68k)

SYNOPSIS

l-cc [-cMpSvkVXmn] [-o *exec***] [-D***var***] [-I***dir***] [-l***name***]**
[*source ...* **] [** *object ...* **] [** *library ...* **]**

DESCRIPTION

`l-cc` is the legacy Lynx proprietary C compiler for x86 and 68k LynxOS systems. It is now used only to build a small part of the LynxOS kernel and libraries that still have Lynx C-style in-line assembly. When we complete the conversion of this code to GNU C-style in-line assembly, we will remove `l-cc` and its related tools from LynxOS. We strongly recommend that you use the GNU C tools (see `gcc(1)`) for your applications.

`l-cc` provides a convenient interface to the C compiler, linker, and disassembler. Arguments whose names end in **.c** are taken to be C language source files, and are translated with the compiler into object files. Object file names are created by replacing the trailing **.c** in the source file names with **.o**. Normally, `l-cc` invokes the linker after all source files are compiled, providing it with the names of all object files created or explicitly listed, along with any other file names present on the `l-cc` invocation line (these are usually libraries). The resulting executable file is called by default `a.out` for historical reasons.

On 68k-based LynxOS systems, `l-cc` passes the option `-D__68K__` to the compiler. On x86-based systems, `l-cc` passes the option `-D__X86__`. In either case, if the `int` data type involves 32-bit values, `l-cc` passes `-DIBITS32` to the compiler.

OPTIONS

-c Compile only. Source files will be compiled but not linked. Any other file arguments are ignored.

–D*var*
–D*var=value*

Define C preprocessor variable *var*, optionally giving a *value*, for each compilation performed. The effect is identical to including a line of the following form at the very top of each source file.:

```
#define var value
```

–I*directory* The named directory will be searched after the current directory "**.**" and before `/usr/include` for the `include` files specified in the source files. Multiple **–I** options can be present. **–I–** informs `l-cc` not to search `/usr/include`.

–k Instructs the linker to produce Common Object File Format (COFF) output. Setting the environment variable `COFFLD` (to anything) has the same effect as the **–k** option.

–l*name* A shorthand method for passing library file names to the linker. `l-cc` forms library names from **–l** options by prepending the name of a library directory and the string `lib` to *name*, and appending **.a** to the result. The usual library directory is `/lib`, though others can be specified with the `LIBPATH` environment variable. Thus, the option **–lloc** causes the string `/lib/libloc.a` to be passed to the linker.

–M Passed directly to the linker, causing a symbolic address map file to be generated along with the executable file. The map file name is created by appending **.m** to the name of the executable file name.

–m Compile and link a multi-threaded program. The default libraries and header files under LynxOS do not provide for the creation of a program that will have multiple asynchronous threads of control operating within the process. In order to create such a program, the **–m** option should be given to `l-cc`. This has the effect of sending `–D_MULTITHREADED` to the compiler and searching for libraries in the multi-threaded library directory `/lib/thread` before the standard library directory `/lib`. Using the multi-threaded libraries and header files necessitates a small amount of overhead, in that static data must be guarded by mutual-exclusion locks and so forth. In addition there is some start-up and tear-down overhead for a multi-threaded process.

-n Don't link in `libc.a` or `init.o` by default. These files can still be linked in if specifically declared.

-o *exec* Allows a name other than `a.out` to be used for the executable file.

-p Causes execution profiling code to be linked into the executable file. When a program so compiled runs and terminates gracefully, a file called `prof.out` is created in the working directory (of the program). The `prof` utility reads the information in the file to produce a table showing how much time each function consumed. The **-p** option implies **-M**.

-S Each source file is compiled and disassembled. File names for the assembly language results are generated by changing the trailing **.c** to **.s** for each source file. The object files are not linked.

-v Instructs the linker to issue a list of object file sizes to standard output. As the linker opens library files, the library file names are printed as well.

ENVIRONMENT VARIABLES

`l-cc` examines each of the following environment variables before beginning the compilation or linking process.

`COMPILER` Gives the name of the executable file to be used to translate C source to object files. If not defined, `l-cc` uses `/l-cc/l-mcc`

`LINKER` Gives the name of the executable file to be used to link object files into an executable file. If not defined, `l-cc` uses `/bin/l-ld`.

`DISASM` Gives the name of the executable file to be used to disassemble object files when the **-S** switch is given. If not defined, `l-cc` uses `/l-cc/l-disasm`.

INCPATH
Gives a list of directories to be searched for include files. The format of INCPATH is:

`directory:directory:... :directory`

Directory names should not include a trailing "/". Each directory so named is searched after directories given with **-I** options, but before the system include file directory. LIBPATH Gives a list of directories to be searched for libraries requested with **-l** options. The format of LIBPATH is the same as that of INCPATH. The directories named are searched before the system library directory.

NO_AOUT
If defined at all, l-cc will use the name of the first source or object file given to form an executable file name by removing the suffix (**.c** or **.o**). Thus, if NO_AOUT is set, the command `l-cc program.c` will create an executable file named program.

COFFLD
This environment variable is not read directly by l-cc, but rather by the linker as a substitute for the linker's **-k** option. Since l-cc passes the **-k** option directly to the linker, setting COFFLD (to anything) has the same effect as giving the **-k** option to l-cc.

DIAGNOSTICS

Error messages from l-cc itself and from all programs it invokes are written to standard error. If a subprogram fails, l-cc exits with whatever value the subprogram used.

SEE ALSO

Utility Programs - l-disasm, l-ld, libr

System Calls - st_build()

Library Functions - pthread_create(), st_new()

File Formats - a.out

NAME

`ld` – the GNU linker

SYNOPSIS

ld [-o *output*] *objfile*...
[-b *input-format*] **[-c** *commandfile*] **[-d|-dc|-dp]**
[-defsym *symbol* **=** *expression*] **[-e** *entry*] **[-embedded-relocs]**
[-F] [-F *format*] **[-format** *input-format*] **[-g] [-G** *size*]
[--help] [-i] [-l*ar*] **[-L***searchdir*] **[-M] [-Map** *mapfile*]
[-m *emulation*] **[-n|-N] [-noinhibit-exec] [-no-keep-memory]**
[-no-init-section] [-keep-memory] [-oformat *output-format*]
[-R *filename*] **[-relax] [-r|-Ur] [-S] [-s] [-sort-common]**
[-split-by-reloc *count*] **[-split-by-file]**
[-T *commandfile*] **[-Ttext** *textorg*] **[-Tdata** *dataorg*]
[-Tbss *bssorg*] **[-t] [-u** *sym*] **[-V] [-v] [--verbose]**
[--version] [-warn-common] [-warn-constructors]
[-warn-once] [--whole-archive] [-X] [-x]

DESCRIPTION

`ld` combines a number of object and archive files, relocates their data and ties up symbol references. Often the last step in building a new compiled program to run is a call to `ld`.

`ld` accepts Linker Command Language files to provide explicit and total control over the linking process. This man page does not describe the command language; see the manual *ld: the GNU linker*, for full details on the command language and on other aspects of the GNU linker.

This version of `ld` uses the general purpose BFD libraries to operate on object files. This allows `ld` to read, combine, and write object files in many different formats—for example, COFF or `a.out`. Different formats may be linked together to produce any available kind of object file. You can use '`objdump -i`' to get a list of formats supported on various architectures; see `objdump`(1).

Aside from its flexibility, the GNU linker is more helpful than other linkers in providing diagnostic information. Many linkers abandon execution immediately upon encountering an error; whenever possible, `ld` continues executing, allowing you to identify other errors (or, in some cases, to get an output file in spite of the error).

The GNU linker `ld` is meant to cover a broad range of situations, and to be as compatible as possible with other linkers. As a result, you have many choices to control its behavior through the command line, and through environment variables.

OPTIONS

The plethora of command-line options may seem intimidating, but in actual practice few of them are used in any particular context. For instance, a frequent use of `ld` is to link standard Unix object files on a standard, supported Unix system. On such a system, to link a file `hello.o`:

```
ld -o output /lib/crt0.o hello.o -lc
```

This tells `ld` to produce a file called `output` as the result of linking the file `/lib/crt0.o` with `hello.o` and the library `libc.a` which will come from the standard search directories.

The command-line options to `ld` may be specified in any order, and may be repeated at will. For the most part, repeating an option with a different argument will either have no further effect, or override prior occurrences (those further to the left on the command line) of an option.

The exceptions—which may meaningfully be used more than once—are **A**, **-b** (or its synonym **-format**), **-defsym**, **-L**, **-l**, **-R**, and **-u**.

The list of object files to be linked together, shown as *objfile*, may follow, precede, or be mixed in with command-line options; save that an *objfile* argument may not be placed between an option flag and its argument.

Usually the linker is invoked with at least one object file, but other forms of binary input files can also be specified with **-l**, **-R**, and the script command language. If *no* binary input files at all are specified, the linker does not produce any output, and issues the message ‘`No input files`’.

Option arguments must either follow the option letter without intervening whitespace, or be given as separate arguments immediately following the option that requires them.

-b *input-format* Specify the binary format for input object files that follow this option on the command line. You don't usually need to specify this, as `ld` is configured to expect as a default input format the most usual format on each machine. *input-format* is a text string, the name of a particular format supported by the BFD libraries. **-format** *input-format* has the same effect, as does the script command `TARGET`.

You may want to use this option if you are linking files with an unusual binary format. You can also use **-b** to switch formats explicitly (when linking object files of different formats), by including **-b** *input-format* before each group of object files in a particular format.

The default format is taken from the environment variable `GNUTARGET`. You can also define the input format from a script, using the command `TARGET`.

-c *commandfile* Directs `ld` to read link commands from the file *commandfile*. These commands will completely override `ld`'s default link format (rather than adding to it); *commandfile* must specify everything necessary to describe the target format.

You may also include a script of link commands directly in the command line by bracketing it between '`{`' and '`}`' characters.

-d
-dc
-dp These three options are equivalent; multiple forms are supported for compatibility with other linkers. Use any of them to make `ld` assign space to common symbols even if a relocatable output file is specified (**-r**). The script command `FORCE_COMMON_ALLOCATION` has the same effect.

-defsym *symbol* = *expression*
Create a global symbol in the output file, containing the absolute address given by *expression*. You may use this option as many times as necessary to define multiple symbols in the command line. A limited form of arithmetic is supported for the *expression* in this context: you may give a hexadecimal constant or the name of an existing symbol, or use + and – to add or subtract hexadecimal constants or symbols. If you need more elaborate expressions, consider using the linker command language from a script.

-e *entry*
Use *entry* as the explicit symbol for beginning execution of your program, rather than the default entry point. for a discussion of defaults and other ways of specifying the entry point.

-F
-F*format*
Some older linkers used this option throughout a compilation toolchain for specifying object-file format for both input and output object files. `ld`'s mechanisms (the **-b** or **-format** options for input files, the `TARGET` command in linker scripts for output files, the `GNUTARGET` environment variable) are more flexible, but it accepts (and ignores) the **-F** option flag for compatibility with scripts written to call the old linker.

-format *input-format*
Synonym for **-b** *input-format*.

-g
Accepted, but ignored; provided for compatibility with other tools.

--help
Print a summary of the command-line options on the standard output and exit. This option and **--version** begin with two dashes instead of one for compatibility with other GNU programs. The other options start with only one dash for compatibility with other linkers.

-i
Perform an incremental link (same as option **-r**).

-l*ar* Add an archive file *ar* to the list of files to link. This option may be used any number of times. `ld` will search its path-list for occurrences of `lib`*ar*.a for every *ar* specified.

-L*searchdir* This command adds path *searchdir* to the list of paths that `ld` will search for archive libraries. You may use this option any number of times.

The default set of paths searched (without being specified with **-L**) depends on what emulation mode `ld` is using, and in some cases also on how it was configured. The paths can also be specified in a link script with the `SEARCH_DIR` command.

-M Print (to the standard output file) a link map—diagnostic information about where symbols are mapped by `ld`, and information on global common storage allocation.

-Map *mapfile* Print to the file *mapfile* a link map—diagnostic information about where symbols are mapped by `ld`, and information on global common storage allocation.

-m *emulation* Emulate the *emulation* linker. You can list the available emulations with the **--verbose** or **-V** options. This option overrides the compiled-in default, which is the system for which you configured `ld`.

-N Specifies readable and writable `text` and `data` sections. If the output format supports Unix style magic numbers, the output is marked as `OMAGIC`.

When you use the **-N** option, the linker does not page-align the data segment.

-n sets the text segment to be read only, and `NMAGIC` is written if possible.

-noinhibit-exec

Normally, the linker will not produce an output file if it encounters errors during the link process. With this flag, you can specify that you wish the output file retained even after non-fatal errors.

-no-init-section
Disables the writing of init sections to the designated target file. This feature was added to make `mkshlib(1)` work properly. It currently is supported only with COFF file formats.

-no-keep-memory
-keep-memory Specify the optimization for memory usage or speed. The **-no-keep-memory** option (default) tells the linker to optimize for memory usage by re-reading the symbol tables as necessary. This is desirable when linking a large executable. The **-keep-memory** option tells the linker to instead optimize for speed by caching the symbol tables of input files in memory.

-o *output* *output* is a name for the program produced by `ld`; if this option is not specified, the name `a.out` is used by default. The script command `OUTPUT` can also specify the output file name.

-oformat *output-format*
Specify the binary format for the output object file. You don't usually need to specify this, as `ld` is configured to produce as a default output format the most usual format on each machine. *output-format* is a text string, the name of a particular format supported by the BFD libraries. The script command `OUTPUT_FORMAT` can also specify the output format, but this option overrides it.

-R *filename* Read symbol names and their addresses from *filename*, but do not relocate it or include it in the output. This allows your output file to refer symbolically to absolute locations of memory defined in other programs.

-r Generates relocatable output—i.e., generate an output file that can in turn serve as input to `ld`. This is often called *partial* linking. As a side effect, in environments that support standard Unix magic numbers, this option also sets the output file's magic number to `OMAGIC`. If this option is not specified, an absolute file is produced. When linking C++ programs, this option *will not* resolve references to constructors; **-Ur** is an alternative.

This option does the same as **-i**.

-S Omits debugger symbol information (but not all symbols) from the output file.

-s Omits all symbol information from the output file.

-sort-common Normally, when `ld` places the global common symbols in the appropriate output sections, it sorts them by size. First come all the one byte symbols, then all the two bytes, then all the four bytes, and then everything else. This is to prevent gaps between symbols due to alignment constraints. This option disables that sorting.

-split-by-reloc *count*
Attempts to create extra sections in the output file so that no single output section in the file contains more than *count* relocations. This is useful when generating huge relocatable for downloading into certain real time kernels with the COFF object file format; since COFF cannot represent more than 65535 relocations in a single section. Note that this will fail to work with object file formats which do not support arbitrary sections. The linker will not split up individual input sections for redistribution, so if a single input section contains more than *count* relocations one output section will contain that many relocations.

-split-by-file
Similar to **-split-by-reloc** but creates a new output section for each input file.

-Tbss *org*
-Tdata *org*
-Ttext *org* Use *org* as the starting address for—respectively—the `bss`, `data`, or the `text` segment of the output file. *textorg* must be a hexadecimal integer.

-T *commandfile* Equivalent to **-c** *commandfile*; supported for compatibility with other tools.

-t Prints names of input files as `ld` processes them.

-u *sym* Forces *sym* to be entered in the output file as an undefined symbol. This may, for example, trigger linking of additional modules from standard libraries. **-u** may be repeated with different option arguments to enter additional undefined symbols.

-Ur For anything other than C++ programs, this option is equivalent to **-r**: it generates relocatable output—i.e., an output file that can in turn serve as input to `ld`. When linking C++ programs, **-Ur** *will* resolve references to constructors, unlike **-r**.

--verbose Display the version number for `ld` and list the supported emulations. Display which input files can and can not be opened.

-v, -V Display the version number for `ld`. The **-V** option also lists the supported emulations.

--version Display the version number for `ld` and exit.

-warn-common Warn when a common symbol is combined with another common symbol or with a symbol definition. Unix linkers allow this somewhat sloppy practice, but linkers on some other operating systems do not. This option allows you to find potential problems from combining global symbols.

-warn-constructors
Warn if any global constructors are used. This is only useful for a few object file formats. For formats like COFF or ELF, the linker can not detect the use of global constructors.

-warn-once Only warn once for each undefined symbol, rather than once per module which refers to it.

--whole-archive
For each archive mentioned on the command line, include every object file in the archive in the link, rather than searching the archive for the required object files. This is normally used to turn an archive file into a shared library, forcing every object to be included in the resulting shared library.

-X	Delete all temporary local symbols. For most targets, this is all local symbols whose names begin with 'L'.
-x	Delete all local symbols.

ENVIRONMENT

You can change the behavior of ld with the environment variable GNUTARGET.

GNUTARGET determines the input-file object format if you don't use **-b** (or its synonym **-format**). Its value should be one of the BFD names for an input format. If there is no GNUTARGET in the environment, ld uses the natural format of the host. If GNUTARGET is set to default then BFD attempts to discover the input format by examining binary input files; this method often succeeds, but there are potential ambiguities, since there is no method of ensuring that the magic number used to flag object-file formats is unique. However, the configuration procedure for BFD on each system places the conventional format for that system first in the search-list, so ambiguities are resolved in favor of convention.

SEE ALSO

Utility Programs -	objdump
Documents -	*ld: the GNU linker*, Steve Chamberlain and Roland Pesch;
	The GNU Binary Utilities, Roland H. Pesch.

COPYING

NAME

`l-disasm` – legacy Lynx object disassembler (x86, 68k, SPARC)

SYNOPSIS

l-disasm [-o] [-f *function***] [-l** *library file***] [-L** *library***]**
[-S *section***]** *file ...*

DESCRIPTION

`l-disasm` is a legacy utility and will be removed in future releases of LynxOS. We recommend that using the GNU `objdump` utility; with `objdump`, you can disassemble objects with the **-d** option. See `objdump(1)` for more information.

`l-disasm` reads the given `a.out` or `coff` format object or executable *file* and writes a translation into assembly language of its contents to standard output. All addresses and address references which can be related to symbols in the symbol table are printed as those symbols. For each instruction, two location values are printed to the left. The first gives the absolute location in the text segment, and the second gives the distance from the most recent label. A sample of `l-disasm` output for 68k LynxOS systems follows:

```
Text size: d70
Data size: 1bc
Bss size: 14
Syms size: 360
Entry at: 0
TRX size: 780
DRX size: 0
.text
_zfree:
00000000 00000000  LINK A6,    #0x0
00000004 00000004  TST.L       8(A6)
00000008 00000008  BEQ.B       0x16
0000000a 0000000a  MOVE.L      8(A6), -(A7)
0000000e 0000000e  JSR         _free.L
```

```
00000014 00000014  ADDQ.L      #4, A7
00000016 00000016  UNLK        A6
00000018 00000018  RTS
_awcon:
0000001a 00000000  LINK A6,     #0xffffffe4
0000001e 00000004  MOVE.L      _trel_len.L, -4(A6)
00000026 0000000c  MOVE.L      _the_trel.L, -8(A6)
0000002e 00000014  MOVE.L      20(A6), D0
00000032 00000018  MOVE.L      _the_text.L, D1
00000038 0000001e  SUB.L       D1, D0
0000003a 00000020  MOVE.L      D0, -16(A6)
0000003e 00000024  MOVEQ.L     #1,D0
00000040 00000026  CMP.L       12(A6), D0
00000044 0000002a  BNE.W       0x11c
00000048 0000002e  MOVE.L      -16(A6), D0
0000004c 00000032  ADD.L       _head_size.L, D0
00000052 00000038  ADD.L       D0, 16(A6)
00000056 0000003c  BRA.W        0x11c
```

OPTIONS

Options specific to x86 and 68k systems:

-o	Print the opcodes also.
-f *function*	Disassemble only the given function.
-l *library file*	The first *file* specified is taken from *library file* rather than from the file system. *library file* must be in `libr` format.
-L *library*	Disassemble all the object files in *library*. *library* must be in `libr` format.
-S *section*	Disassemble only the specified *section*.

For SPARC systems only:

-e	Dump, starting at the offset given by the `a.out` header at the entry point. The default is to begin dumping directly after the `a.out` header.
-s	Dump symbol table.
-S	Dump strings.

-? Print a usage message.

NOTE

`l-cc` uses `l-disasm` when invoked with the **-S** option.

SEE ALSO

Utility Programs - `objdump, l-cc, st`

NAME

`less` – interactive paginator files, viewing

SYNOPSIS

`less [ –[+]aABcCdeEimMnqQuUsw] [–b` *n* **`] [–h`** *n* **`] [–x`** *n* **`] [–[z]`** *n* **`]`**

DESCRIPTION

`less` is a program which might best be described as a read-only text editor, allowing forward and backwards movement through files. `less` does not have to read the entire input file before starting, so with large input files it starts up faster than most text editors. `less` uses `termcap` (or `terminfo` on some systems), so it can run on a variety of terminals. There is even limited support for hardcopy terminals. (On a hardcopy terminal, lines that should be printed at the top of the screen are prefixed with an up-arrow.)

Commands are based on those of both the UNIX `more` program and `vi`. Commands may be preceded by a decimal number, called *n* in the descriptions below. The number is used by some commands, as indicated.

COMMANDS

In the following descriptions, `^X` means control-X. `ESC` stands for the `ESCAPE` key. For example, `ESC-v` means the two-character sequence `ESCAPE` key, then `v`:

Character	Meaning
`H`	Help: display a summary of these commands. If you forget all the other commands, remember this one.

`SPACE`
`f`
`^F`
`^V`
	Scroll forward n lines, default one window (see option `-z` below). If *n* is more than the screen size, only the final screenful is displayed. Warning: some systems use `^V` as a special literalization character.

`b`
`^B`
`ESC-v`
	Scroll backward *n* lines, default one window (see option `-z` below). If *n* is more than the screen size, only the final screenful is displayed.

`RETURN`
`^N`
`e`
`^E`
`j`
`^J`
	Scroll forward *n* lines, default 1. The entire *n* lines are displayed, even if *n* is more than the screen size.

`y`
`^Y`
`^P`
`k`
`^K`
	Scroll backward *n* lines, default 1. The entire *n* lines are displayed, even if *n* is more than the screen size. Warning: by default LynxOS uses `^Y` as the suspend job control character. To use `^Y` the suspend character must be remapped.

`d` `^D`	Scroll forward *n* lines, default one half of the screen size. If *n* is specified, it becomes the new default for subsequent *d* and *u* commands.
`u` `^U`	Scroll backward *n* lines, default one half of the screen size. If *n* is specified, it becomes the new default for subsequent *d* and *u* commands.

`r`
`^R`
`^L`
	Repaint the screen.

Character	Meaning
`R`	Repaint the screen, discarding any buffered input. Useful if the file is changing while it is being viewed.

`g`
`<`
`ESC-<`
	Go to line *n* in the file, default 1 (beginning of file). (Warning: this may be slow if *n* is large.)

`G`
`>`
`ESC->`
	Go to line *n* in the file, default the end of the file. (Warning: this may be slow if *n* is large, or if n is not specified and standard input, rather than a file, is being read.)

`p` `%`	Go to a position *n* percent into the file. *n* should be between 0 and 100. (This works if standard input is being read, but only if `less` has already read to the end of the file. It is always fast, but not always useful.)
`m`	Followed by any lower case letter, marks the current position with that letter.
`'` `^X^X`	Followed by any lower case letter, returns to the position which was previously marked with that letter. Followed by another single quote, returns to the position at which the last "large" movement command was executed. All marks are lost when a new file is examined.
`/` *pattern*	Search forward in the file for the *n*th line containing the pattern. If omitted, *n* defaults to 1. The pattern is a regular expression, as recognized by `grep`. The search starts at the second line displayed (but see the `-a` option, which changes this).
`?` *pattern*	Search backward in the file for the *n*th line containing the pattern. The search starts at the line immediately before the top line displayed.
`/!` *pattern*	Like `/`, but the search is for the *n*th line that does not contain the pattern.
`?!` *pattern*	Like `?`, but the search is for the *n*th line that does not contain the pattern.
`n`	Repeat previous search, for *n*th line containing the last pattern (or not containing the last pattern, if the previous search was `/!` or `?!`).
`E` *filename*	Examine a new file. If *filename* is missing, the "current" file (see the `N` and `P` commands below) from the list of files in the command line is re-examined. If *filename* is a pound sign (`#`), the previously examined file is re-examined.

Character	Meaning
^X^V :e	Same as E. Warning: some systems use ^V as a special literalization character.
N :n	Examine the next file (from the list of files given in the command line). If a number *n* is specified, the *n*th next file is examined.
P :p	Examine the previous file. If a number *n* is specified, the *n*th previous file is examined.
= ^G	Prints some information about the file being viewed, including its name and the line number and byte offset of the bottom line being displayed. If possible, it also prints the length of the file and the percent of the file above the last displayed line.
–	Followed by one of the command line option letters (see below), this will change the setting of that option and print a message describing the new setting. If the option letter has a numeric value (such as –b or –h), or a string value (such as –P or –t), a new value may be entered after the option letter.
_	(Underscore.) Followed by one of the command line option letters (see below), this will print a message describing the current setting of that option. The setting of the option is not changed.
+ *cmd*	Causes the specified *cmd* to be executed each time a new file is examined. For example, +G causes less to initially display each file starting at the end rather than the beginning.
V	Prints the version number of less being run.
q :q ZZ	Exits less.
The following two commands may or may not be valid, depending on your installation.	
v	Invokes an editor to edit the current file being viewed. The editor is taken from the environment variable EDITOR, or defaults to vi.
! *command*	Invokes a shell to run the command given. A percent sign in the command is replaced by the name of the current file. !! repeats the last shell command. ! with no shell command simply invokes a shell. In all cases, the shell is taken from the environment variable SHELL, or defaults to sh.

OPTIONS

Command line options are described below. Most options may be changed while `less` is running, via the – command.

Options are also taken from the environment variable `LESS`. The environment variable is parsed before the command line, so command line options override the `LESS` environment variable. If an option appears in the `LESS` variable, it can be reset to its default on the command line by beginning the command line option with **-+**.

A dollar sign ($) may be used to signal the end of an option string. This is important only for options like **-P** which take a following string.

-a Normally, forward searches start just after the top displayed line (that is, at the second displayed line). Thus, forward searches include the currently displayed screen. The **-a** option causes forward searches to start just after the bottom line displayed, thus skipping the currently displayed screen.

-A The **-A** option causes searches to start at the second SCREEN line displayed, as opposed to the default which is to start at the second REAL line displayed. For example, suppose a long real line occupies the first three screen lines. The default search will start at the second real line (the fourth screen line), while the **-A** option will cause the search to start at the second screen line (in the midst of the first real line). (This option is rarely useful.)

-b*n* The **-b** option tells `less` to use a non-standard number of buffers. Buffers are 1K, and normally 10 buffers are used (unless data is coming from standard input; see the **-B** option). The number *n* specifies a different number of buffers to use.

-B Normally, when data is coming from standard input, buffers are allocated automatically as needed, to avoid loss of data. The **-B** option disables this feature, so that only the default number of buffers are used. If more data is read than will fit in the buffers, the oldest data is discarded.

-c Normally, `less` will repaint the screen by scrolling from the bottom of the screen. If the **-c** option is set, when `less` needs to change the entire display, it will paint from the top line down.

-C The **-C** option is like **-c**, but the screen is cleared before it is repainted.

-d Normally, `less` will complain if the terminal is dumb; that is, lacks some important capability, such as the ability to clear the screen or scroll backwards. The **-d** option suppresses this complaint (but does not otherwise change the behavior of the program on a dumb terminal).

-e Normally the only way to exit `less` is via the `q` command. The **-e** option tells `less` to automatically exit the second time it reaches end-of-file.

-E The **-E** flag causes `less` to exit the first time it reaches end-of-file.

-h Normally, `less` will scroll backwards when backwards movement is necessary. The **-h** option specifies a maximum number of lines to scroll backwards. If it is necessary to move backwards more than this many lines, the screen is repainted in a forward direction. (If the terminal does not have the ability to scroll backwards, **-h0** is implied.)

-i The **-i** option causes searches to ignore case; that is, upper case and lower case are considered identical. Also, text which is overstruck or underlined can be searched for.

-l The **-l** option, followed immediately by a filename, will cause `less` to copy its input to the named file as it is being viewed. This applies only when the input file is a pipe, not an ordinary file. If the file already exists, `less` will ask for confirmation before overwriting it.

-L The **-L** option is like **-l**, but it will overwrite an existing file without asking for confirmation. If no log file has been specified, the **-l** and **-L** options can be used from within `less` to specify a log file. Without a file name, they will simply report the name of the log file.

-m Normally, `less` prompts with a colon. The **-m** option causes `less` to prompt verbosely with the percentage into the file.

-M The **-M** option causes `less` to prompt even more verbosely than **-m**.

-n The **-n** flag suppresses line numbers. The default (to use line numbers) may cause `less` to run more slowly in some cases, especially with a very large input file. Suppressing line numbers with the **-n** flag will avoid this problem. Using line numbers means: the line number will be displayed in the verbose prompt and in the = command, and the `v` command will pass the current line number to the editor.

-P The **-P** option provides a way to tailor the three prompt styles to your own preference. This option is normally placed in the `LESS` environment variable, rather than with the invocation of `less` (although that is of course allowed). Such an option must either be the last option in the `LESS` variable, or be terminated by a dollar sign. **-P** followed by a string changes the default (short) prompt to that string. **-Pm** changes the medium (**-m**) prompt to the string, and **-PM** changes the long (**-M**) prompt. Also, **-P=** changes the message printed by the = command to the given string. All prompt strings consist of a sequence of letters and special escape sequences. See the section on `PROMPTS` for more details.

-q Normally, if an attempt is made to scroll past the end of the file or before the beginning of the file, the terminal bell is rung to indicate this fact. The **-q** option tells `less` not to ring the bell at such times. If the terminal has a "visual bell", it is used instead.

-Q Even if **-q** is given, `less` will ring the bell on certain other errors, such as typing an invalid character. The **-Q** option tells `less` to be quiet all the time; that is, never ring the terminal bell. If the terminal has a “visual bell”, it is used instead.

-s The **-s** option causes consecutive blank lines to be squeezed into a single blank line.

-t The **-t** option, followed immediately by a `TAG`, will edit the file containing that tag. For this to work, there must be a file called `tags` in the current directory, which was previously built by the `ctags` command. This option may also be specified from within `less` (using the **-** command) as a way of examining a new file.

-u If the **-u** option is given, backspaces are treated as printable characters; that is, they are sent to the terminal when they appear in the input.

-U If the **-U** option is given, backspaces are printed as the two character sequence `^H`. If neither **-u** nor **-U** is given, backspaces that appear adjacent to an underscore character are treated specially: the underlined text is displayed using the terminal’s hardware underlining capability. Also, backspaces that appear between two identical characters are treated specially: the overstruck text is printed using the terminal's hardware boldface capability. Other backspaces are deleted, along with the preceding character.

-w Normally, `less` uses a tilde character to represent lines past the end of the file. The **-w** option causes blank lines to be used instead.

-x*n* The **-x***n* option sets tab stops every *n* positions. The default for *n* is 8.

-z When given a backwards or forwards window command, `less` will by default scroll backwards or forwards one screenful of lines. The **-z***n* option changes the default scrolling window size to *n* lines. Note that the **z** is optional for compatibility with the UNIX program `more`.

+	If a command line option begins with **+**, the remainder of that option is taken to be an initial command to `less`. For example, **+** tells `less` to start at the end of the file rather than the beginning, and **+/xyz** tells it to start at the first occurrence of **xyz** in the file. As a special case, **+number** acts like **+numberg**; that is, it starts the display at the specified line number (however, see the caveat under the **g** command above). If the option starts with **++**, the initial command applies to every file being viewed, not just the first one. The **+** command described previously may also be used to set (or change) an initial command for every file.

KEY BINDINGS

You may define your own `less` commands by using the program `lesskey` to create a file called `.less` in your home directory. This file specifies a set of command keys and an action associated with each key. See the `lesskey` manual page for more details.

PROMPTS

The **-P** option allows you to tailor the prompt to your preference. The string given to the **-P** option replaces the specified prompt string. Certain characters in the string are interpreted specially. The prompt mechanism is rather complicated to provide flexibility, but the ordinary user need not understand the details of constructing personalized prompt strings.

A percent sign followed by a single character is expanded according to what the following character is:

`%b` *X*	Replaced by the byte offset into the current input file. The b is followed by a single character (shown as *X*) that specifies the line whose byte offset is to be used. If the character is a `t` the byte offset of the top line in the display is used, an `m` means use the middle line, a `b` means use the bottom line, and a `B` means use the line just after the bottom line.
`%f`	Replaced by the name of the current input file.

`%i`	Replaced by the index of the current file in the list of input files.
`%l` *X*	Replaced by the line number of a line in the input file. The line to be used is determined by the *X*, as with the `%b` option.
`%m`	Replaced by the total number of input files.
`%p` *X*	Replaced by the percent into the current input file. The line used is determined by the *X* as with the `%b` option.
`%s`	Replaced by the size of the current input file.
`%t`	Causes any trailing spaces to be removed. Usually used at the end of the string, but may appear anywhere.
`%x`	Replaced by the name of the next input file in the list.

If any item is unknown (for example, the file size if input is a pipe), a question mark is printed instead.

The format of the prompt string can be changed depending on certain conditions. A question mark followed by a single character acts like an `IF`; depending on the following character, a condition is evaluated. If the condition is true, any characters following the question mark and condition character, up to a period, are included in the prompt. If the condition is false, such characters are not included. A colon appearing between the question mark and the period can be used to establish an `ELSE`; any characters between the colon and the period are included in the string if and only if the `IF` condition is false.

Condition characters (which follow a question mark) may be:

`?a`	True if any characters have been included in the prompt so far.
`?b` *X*	True if the byte offset of the specified line is known.
`?e`	True if at end-of-file.
`?f`	True if there is an input filename (that is, if input is not a pipe).
`?l` *X*	True if the line number of the specified line is known.

`?m`	True if there is more than one input file.
`?n`	True if this is the first prompt in a new input file.
`?p` *X*	True if the percent into the current input file of the specified line is known.
`?s`	True if the size of the current input file is known.
`?x`	True if there is a next input file (that is, if the current input file is not the last one).

Any characters other than the special ones (question mark, colon, period, percent, and backslash) become literally part of the prompt. Any of the special characters may be included in the prompt literally by preceding it with a backslash.

Some examples:

```
?f%f:Standard input.
```

This prompt prints the filename, if known; otherwise the string “Standard input”.

```
?f%f .?ltLine %lt:?pt%pt\%:?btByte %bt:-...
```

This prompt would print the filename, if known. The filename is followed by the line number, if known, otherwise the percent if known, otherwise the byte offset if known. Otherwise, a dash is printed. Notice how each question mark has a matching period, and how the `%` after the `%pt` is included literally by escaping it with a backslash.

```
?n?f%f\ .?m(file\ %i\ of\ %m)\ ..?e(END)\ ?x-\ Next\\:\ %x..%t
```

This prints the filename if this is the first prompt in a file, followed by the “file *n* of *n*” message if there is more than one input file. Then, if we are at end-of-file, the string “(END)” is printed followed by the name of the next file, if there is one. Finally, any trailing spaces are truncated. This is the default prompt. For reference, here are the defaults for the other two prompts (**-m** and **-M** respectively). Each is broken into two lines here for readability only.

```
?n?f%f\ .?m(file\ %i\ of\ %m)\ ..?e(END)\ ?x-\ Next\\:\ %x.:
?pB%pB\\%:byte\ %bB?s\%s...%t
```

```
?f%f\ .?n?m(file\ %i\ of\ %m)\ ..?ltline\ %lt\:byte\ %bB?s/%s\
.. ?e(END)\ ?x-\ Next\\:\ %x.:?pB%pB\\%..%t
```

And here is the default message produced by the = command:

```
?f%f\ .?m(file\ %i\ of\ %m)\ .?ltline\ %lt\ . byte\ %bB?s/%s.\
?e(END)\ :?pB%pB\\%..%t
```

WARNINGS

The **=** command and prompts (unless changed by **-P**) report the line number of the line at the top of the screen, but the byte and percent of the line at the bottom of the screen.

SEE ALSO

Utility Programs - `lesskey`

NAME

lesskey – specify key bindings for less

SYNOPSIS

lesskey [-o *output***] [** *input* **]**

DESCRIPTION

lesskey is used to specify a set of key bindings to be used by less. The input file is a text file that describes the key bindings, and the output file is a binary file that is used by less. If no input file is specified, stdin is used. If no output file is specified, $HOME/.less is used.

The input file consists of lines of the form:

string action

string and *action* should be separated by one or more spaces or tabs. The *string* is the command key(s) which invoke the action. The *string* may be a single command key, or a sequence of up to 15 keys. The *action* is the name of the less action, from the list below. The characters in the *string* may appear literally, or be prefixed by a caret to indicate a control key. A backslash may be used to cause the following character to be taken literally.

Characters that must be preceded by backslash include caret, space, tab, and the backslash itself. A backslash followed by one, two, or three octal digits may be used to specify a character by its octal value. Blank lines and lines that start with a pound sign (#) are ignored.

As examples of defaults, the table “Default Commands Used By less” describes the set of default command keys used by `less`:

Table 1: Default Commands Used By `less`

Default Command	Description	Default Command	Description
`k`	back-line	`G`	goto-end
`y`	back-line	`>`	goto-end
`^K`	back-line	`\33>`	goto-end
`^Y`	back-line	`g`	goto-line
`^P`	back-line	`<`	goto-line
`b`	back-screen	`\33<`	goto-line
`^B`	back-screen	`’`	goto-mark
`\33v`	back-screen	`^X^X`	goto-mark
`u`	back-scroll	`H`	help
`^U`	back-scroll	`N`	next-file
`?`	back-search	`%`	percent
`E`	examine	`P`	percent
`^X^V`	examine	`p`	prev-file
`+`	first-cmd	`q`	quit
`e`	forw-line	`ZZ`	quit
`j`	forw-line	`^L`	repaint
`^E`	forw-line	`^R`	repaint
`^J`	forw-line	`r`	repaint
`^M`	forw-line	`R`	flush-repaint
`^N`	forw-line	`n`	repeat-search
`f`	forw-screen	`m`	set-mark
`^F`	forw-screen	`!`	shell
`’`	forw-screen	`=`	status

Table 1: Default Commands Used By less

Default Command	Description	Default Command	Description
`\\40`	forw-screen	`^G`	status
`^V`	forw-screen	`-`	toggle-option
`d`	forw-scroll	`_`	display-option
`^D`	forw-scroll	`V`	version
`/`	forw-search	`v`	visual

Commands specified by `lesskey` take precedence over the default commands. A default command key may be disabled by including it in the key file with the action “invalid”.

SEE ALSO

Utility Programs - `less`

NAME

`libr, ar, ranlib` – manipulate object library files

SYNOPSIS

`libr` *cmd library file file ...*

`ar` *cmd library file file ...*

`ranlib -t` *library*

DESCRIPTION

`libr` creates and manipulates libraries of files. A `library` is a single file consisting of the data from one or more other files, along with headers for each original file giving file name and other information. `libraries` are most often used to collect object code modules together for use in software development.

Unless explicitly suppressed, `libr` will generate a table of contents for all global symbols defined in the files contained in the `library`. The table of contents is prepended to the `library` as a separate component called `__.SYMDEF`. Symbols are only extracted from object module files; if the `library` contains no object files, the generation of `__.SYMDEF` should be disabled.

`ar` functions in the same way as `libr`, except that `ar` implicitly suppresses generation or update of the `__.SYMDEF` entry in the library.

The function of `ranlib` is to create or update the `__.SYMDEF` entry in the library. `ranlib` takes a single option **`-t`** that causes the time stamp of the `__.SYMDEF` file to be updated without actually regenerating the symbol table file.

OPTIONS

`libr` and `ar` operate on libraries according to the **`cmd`** given. A **`cmd`** must include one of the following key letters:

`d`	Delete each *file* given from *library*. No complaints are issued if files mentioned are not found in the library.

q	Quickly appends each *file* to the end of the library without examining the library at all. Useful when the files are known to be absent from the library.
r	Replace the version of each *file* in *library* with the current version. If any files mentioned are not in the library already, they are added to the end.
t	Produce a table of contents of *library*. The information is written to standard output. Any *file* arguments are ignored.
u	Similar to **r**, except that each *file* is only replaced if the modification date of the version in the library is older than that of the actual file.
x	Each *file* is extracted from *library* into the current working directory. Note that this option does not remove the file from the library.
s	If a symbol table (`__.SYMDEF`) file is present, all symbols are listed with the names of the corresponding files in the library. If *file* arguments are present, only symbols from those files are printed.

Each of the above command keys can be modified by one or more of the following options.

c	Suppress the comment printed when `libr` creates a new library file during a replace **r**, update **u**, or quick append **q** operation. (This is included mainly for compatibility with other librarians; its utility is in question.)
v	Verbose output. For the **d**, **r**, **u**, and **x** switches, the **v** modifier causes a confirmation for each *file* argument operated upon. For the **t** switch, the **v** modifier produces complete information about each *file* in *library*, instead of just the name.
R	Suppress generation of the `__.SYMDEF` entry in the library.

NOTE

`libr` performs all updates except quick appends by creating a copy of the old library and moving files into it. Therefore enough space should be available to hold two copies of the library.

SEE ALSO

Utility Programs - `cc, tar`

NAME

`lipcrm` – Remove POSIX.1b interprocess communication objects

SYNOPSIS

`lipcrm` *name* **[***name* ...**]**

DESCRIPTION

`lipcrm` unlinks the named POSIX.1b interprocess communication (IPC) object in the system. If there are no processes using the object, then its storage is freed and its resources are returned to the system. POSIX.1b IPC facilities include shared memory, semaphores (both named and unnamed) and message queues. Unnamed semaphores can not be removed by this command.

DIAGNOSTICS

Permission to remove a POSIX.1b IPC object is restricted to either the user with the same UID as object, or root (UID 0).

SEE ALSO

Utility Programs - `lipcs`, `ps`

System Calls - `shm_open()`, `shm_unlink()`

Library Functions - `sem_open()`, `sem_unlink()`, `mq_open()`, `mq_unlink()`

NAME

`lipcs` – list POSIX.1b interprocess communication objects

SYNOPSIS

lipcs [–amqsv]

DESCRIPTION

`lipcs` shows the status of all named POSIX.1b interprocess communication (IPC) facilities on the system. POSIX.1b IPC facilities include shared memory, semaphores (both named and unnamed) and message queues. Unnamed semaphores are not shown by this command.

The output format looks like this (for two levels of verbosity, i.e. **–vv**):

```
message-queue-name         MQ  rw-rw-r--  uid 0  gid 0
    Pid 20 holds 1 refs

__mq-564738512-shm         SHM rw-rw-r--  uid 0  gid 0
    Pid 20 holds 2 refs

unlinked-semaphore-name    SEM rw-r--r--  uid 323  gid 21
*** Unlinked
    Pid 20 holds 3 refs.

shared-memory-region-name  SHM rw-rw-rw- uid 3  gid 2
    Pid 20 holds 1 refs.
    Pid 16 holds 2 refs.
```

For one level of verbosity, it looks like this:

```
message-queue-name         MQ rw-rw-r--  uid 0  gid 0
    Pid 20 holds 1 refs.

__mq-564738512-shm         SHM rw-rw-r-- uid 0  gid 0
    Pid 20 holds 2 refs.

shared-memory-region-name  SHM rw-rw-rw- uid 3  gid 2
    Pid   20 holds  1 refs.
    Pid   16 holds  2 refs.
```

And for no extra verbosity, the output looks like this:

```
message-queue-name          MQ rw-rw-r--  uid 0   gid 0
__mq-564738512-shm          SHM rw-rw-r-- uid 0   gid 0
shared-memory-region-name   SHM rw-rw-rw- uid 3   gid 2
```

The first field is the name of the IPC object. Note that the system creates and uses shared memory for the message queue implementation, and these shared memory regions will appear with two leading underscores in the name. It is recommended that the user not create IPC objects with two leading underscores in the name.

OPTIONS

–a Show all POSIX.1b IPCs currently active. (This is the default)

–m Show only POSIX.1b shared memory segments.

–q Show only POSIX.1b message queues.

–s Show only POSIX.1b named semaphores.

–v Increase the "verbosity" of the output. One **–v** will display information about which process holds references to each object. Two **–v** (could also be **–vv**) will show information about unlinked objects that still have references to them. More than two turns provides information about empty slots in the POSIX.1b IPC name table, objects whose type does not match the type specified on the command line (for the **-[mqs]** switches), and other minutiae.

FILES

/dev/mem	The memory device used to get information from the kernel.

SEE ALSO

Utility Programs - lipcrm, ps

System Calls - shm_open()

Library Functions - sem_open(), mq_open()

NAME

`l-ld` - legacy Lynx object file linker (x86 and 68k)

SYNOPSIS

l-ld [-M] [-V] [-k [*initsecname* **]] [-r] [-v] [-c] [-j]**
[-e *entrypoint*] **[-B** *addr*] **[-D** *addr*] **[-T** *addr*] **[-b** *addr*]
[-d *addr*] **[-t** *addr*] **[-P** *pagesize*] **[-y** *sym*] **[-g] [-o** *name*]
file ... **[-l** *x*] ...

DESCRIPTION

`l-ld` is the legacy Lynx proprietary object file linker for x86 and 68k LynxOS systems. `l-ld` is used to link the LynxOS kernel on these platforms because the kernel still uses some Lynx C-style in-line assembly. When we complete the conversion of this code to GNU C-style in-line assembly, we will remove `l-cc`, `l-ld` and related tools from LynxOS. We strongly recommend that you use the GNU C tools (see `gcc(1)` and `ld(1)`) for your applications.

`l-ld` combines sections and resolves references between the given object and/or archive files to produce a single object file result. The output from `l-ld` is placed in *name*, or in `a.out` if no *name* is given. `l-ld` is capable of producing both `a.out` and COFF executables and can read the following types of files:

- `a.out` object files
- COFF object files
- `a.out` and COFF object files from `libr` archives, and COFF object files from ar (System V) archives.

OPTIONS

-M A map file, giving values for all global symbols, is produced in `name.m` or `a.out.m`.

-V Make the executable that is produced a “System V” executable, as interpreted by the operating system. This is done by writing `ZVMAGIC` (0414) into the executable, or, if the **-k** option is also given, by writing `COFFMAGIC`

(0514) into the executable. The **-V** option has no effect when the **-r** option is also given, because in that case, the output is not an executable.

-k [*initsecname*] Produce COFF output. If the output is an executable (i.e., the **-r** option is not given), then its magic number will be `ZCOFFMAGIC` (0415), unless the **-V** option is also given, in which case the magic number will be `COFFMAGIC` (0514). In addition to producing COFF output, the linker will combine all `.init` sections that it encounters and, if the **-r** option is not given, add them to the beginning of the resulting `.text` section in the output.

If the **-r** option is given, the combined `.init` sections will be made into a separate `.init` section, and this section will be made the first section in the output file. If `initsecname` is given, then the `.text` section of the output (after any `.init` sections have been added) will be renamed to `initsecname`. Renaming the `.text` section in this way is useful in conjunction with the **-r** option to produce object files with `.init` sections (`initsecname` = `.init`). If a space is given between the **-k** and `initsecname`, then `initsecname` should begin with "." to differentiate it from a file name.

-r Forces relocation information for all global symbols to be included in the output. This option allows the linker to produce an object file that is not completely resolved, for use in subsequent linking. Necessary when creating dynamically-loadable device drivers.

-v Verbose. Causes sizes and name of each object file linked to be printed, including files extracted from libraries.

-j Causes the linker to ignore any `NOLOAD` section (referring to the `TEXT` or `DATA` of a shared library) if the absolute address of that section is the same as that of a previously encountered `NOLOAD` section of the same type (`TEXT` or `DATA`). Without the **-j** option, such an occurrence causes the linker to generate a fatal error and abort the linking. When the linker ignores a `NOLOAD` section (under the **-j** option), it prints a warning message, and if the two coincident NOLOAD sections differ in their names as well, the linker prints an additional warning message.

-B *addr* — Start the BSS section at virtual address *addr*. Without the **-B** option, the BSS section will immediately follow the last DATA location.

-b *addr* — Start the BSS section at virtual address *addr*. Without the **-b** option, the BSS section will immediately follow the last DATA location.

-D *addr* — Start the DATA section at virtual address *addr*. Without the **-D** option, the DATA section will start at the page boundary following the TEXT section.

-d *addr* — Start the DATA section at virtual address *addr*. Without the **-d** option, the DATA section will start at the page boundary following the TEXT section.

-T *addr* — Start the TEXT section at virtual address *addr*. Without the **-T** option, the TEXT section will begin at zero.

-t *addr* — Start the TEXT section at virtual address *addr*. Without the **-t** option, the TEXT section will begin at zero.

-P *pagesize* — Causes the linker to assume a system page size of *pagesize*. When the **-P** option is not given, the system page size is used to determine the page boundary following the TEXT section where the DATA section begins.

-g — Include debugging information in the output. l-ld can only copy debugging information from COFF to COFF or from a.out to a.out. In the case of COFF, l-ld retains standard line number debugging information from any input COFF object files and copies it to the output COFF file. The gdb debugger does not support l-ld debugging information.

-l *x* — Link with the library /lib/libx.a, or, if the LIBPATH environment variable is set, with the library $LIBPATH/libx.a. The **-l** option may be specified multiple times.

–o *name*	Output will be written to *name* instead of the default `a.out`.
–y *sym*	Adds *sym* to the list of traced symbols. Each reference and the definition of each traced symbol will be reported. Each of the **–B**, **–D**, and **–T** options can also be given in lower case (**–b**, **–d**, and **–t**). This is so that they can be passed to `l-ld` via `l-cc`, because the capitalized options can have their own meanings for `l-cc`.
–e *entrypoint*	Set the entry point address of the output executable to *entrypoint*, if numeric, or to the address of the symbol *entrypoint*, if non-numeric.
–c	Ignore all `.init` sections. This option is used by the `mkshlib` utility when it is linking the target shared library.

ENVIRONMENT VARIABLES

The following environment variables modify the behavior of `l-ld`:

`LIBPATH`	Causes the linker to search for the library `lib`*x*`.a` in the directory whose name is stored in `LIBPATH`. See the **–l** *x* option.
`COFFLD`	This environment variable functions as a substitute for the **–k** option. If it is set, the linker will behave as if the **–k** option had been given. This is useful where `l-ld` is called from makefiles or shell scripts.

NOTE

The `l-ld` utility is capable of producing several types of output based upon the presence/absence of the three options **–r**, **–k**, and **–V**:

–r –k	Linker writes magic number `COFFMAGIC` (0514) into the output. Output is a non-executable COFF object file.
–r (no **–k**)	Linker writes magic number `OMAGIC` (0407) into the output. Output is a non-executable `a.out` object file.

-k -V (no **-r**)	Linker writes magic number `COFFMAGIC` (0514) into the output. Output is an executable COFF object file that is to be run as a System V program.
-k (no **-r** or **-V**)	Linker writes magic number `ZCOFFMAGIC` (0415) into the output. Output is an executable COFF object file.
-V (no **-r** or **-k**)	Linker writes magic number `ZVMAGIC` (0414) into the output. Output is an executable `a.out` object file that is to be run as a System V program.
(no **-r**, **-k**, or **-V**)	Linker writes magic number `ZMAGIC` (0413) into the output. Output is an executable `a.out` object file.

If the **-k** option is not given and the linker encounters an `.init`, `.lib`, or `NOLOAD` section, it will print a warning message stating that the **-k** option (COFF output) should probably have been given, but proceeds to produce an `a.out` format file anyway with all of the `.init` sections combined and added to the beginning of the `.text` section and all of the `.lib` and `NOLOAD` sections ignored.

If the **-e** option is not given, `l-ld` determines the entry point for an output executable by looking for one of the following symbols, with priority based on their listed order, and sets the entry point to the address of that symbol:

```
_ _start
_start
_ _main
_main
main
```

If none of the above symbols are found, then `l-ld` will set the entry point equal to the beginning of the `.text` section. In cases where none of the above symbols are found and there were `.init` sections added to the beginning of the `.text` section, `l-ld` will set the entry point equal to the point immediately following the added `.init` sections.

`l-ld` is generally used via `l-cc`.

SEE ALSO

Utility Programs - `l-cc, librr, dump, mkshlib`

NAME

`l-mkromk` – legacy utility for creating LynxOS kernel images

SYNOPSIS

l-mkromk [-?epsSv] [-j *baseaddr* **] [-f** *reboot flags* **]**
[-r *rootdev* **] [-O** *osfile* **] [-d** *rootdir* **] [-b** *extrablocks* **]**
[-i *extrainodes* **] [-R** *romstart* **] [-B** *bootcode* **] [-x** *oft* **]**
[-L *loadaddr* **][-E** *entry* **]** *outfile*

DESCRIPTION

`l-mkromk` is the legacy version of the utility used to create images of the LynxOS kernel. This utility has been replaced by `mkimage`, a re-designed version with a greatly improved user interface. We recommend that you use `mkimage` to create rommable kernel images. See `mkimage(1)` and `mkimage.spec(5)` for more information.

`l-mkromk` creates a romable kernel image file. The romable kernel image can be stored in ROM for `rom` booting or kept on a remote server for `tftp` booting. The image consists of `romboot` (optional), LynxOS kernel and LynxOS file system with a basic set of initial files in it. The file system appended at the end is accessed by LynxOS RAM disk driver as root file system. Optionally, in order to save RAM space, the text segments of all executables are stripped and written at the end of the romable kernel image file starting on a page boundary. Rest of the executable file is written with a new magic number and additional information in the beginning, which the loader can recognize and locate the text segments for execution.

-x option should only be specified for `rom` booting; that is, when image file will sit in ROM. Booting from remote server is done by `tftp` code sitting on host ROM; boot code is not part of the image itself. For `romboot`, specifically which segments should be moved to RAM on startup is selected by **-x** option.

OPTIONS

-?	Displays help message of all available options.
-B romboot	Link romboot code with kernel image in the beginning.

-R *romstart* Start address of ROM/RAM space where romable kernel image will initially sit.

-x *oft* Set the execution environment of romable kernel image. If **-x** is not specified, everything is moved to ram on startup.

-xo specifies execute kernel from ROM.

-xf specifies root file system should stay in ROM.

-xt specifies stripped texts should execute from ROM.

-j *baseaddr* Specify end of standard memory (i.e., below 1MB) on an x86 type machine where there is a gap in RAM. This is likely to be `0xA0000` (640k). On systems having contiguous memory this option should not be specified.

-f *reboot flags* Pass Boot flags to kernel on startup. All boot flags are more completely defined in the reboot system call documentation

-r *rootdev* Device ID of root file system. Normally root file system ID is determined automatically by `l-mkromk` by looking at node `/dev/rd0` on the target root file system. This option should only be specified if alternate root ID is desired. Device ID is calculated from `major` and `minor` device IDs as follows:

```
device_id = major * 256 + minor
```

-s Strip text segments from executables.

-S Generate byte swapped file system. By default `l-mkromk` will use big-endian byte ordering. Thus this option is not required if target machine is 680XX based.

-v Verbose mode shows the progress of this utility.

-O *osfile* Name of LynxOS file to run.

-d *rootdir* Normally, to run this utility, one should create an image of all files to be put into the file system and specify the directory name under which this is image is created as the *rootdir*. This instructs `l-mkromk` to map this as the root directory on the file system and copy all the files under it.

-b *extrablocks* Extra number of blocks which the file system should contain. If omitted, `l-mkromk` will create minimal file system, containing just enough number of blocks to hold all the initial set of files to be created on it.

-i *extrainodes* Extra number of inodes which the file system should contain. If omitted, `l-mkromk` will create minimal file system, containing just enough inodes for the initial set of files to be created on it.

-e Expand kernel `bss` in the output (romable kernel image) file.

-L *loadaddr* Physical address in RAM where kernel is loaded. By default this is 0x1000.

-E *entry* Physical entry point for kernel. By default it is physical load address plus kernel header size.

At the end, `l-mkromk` will summarize the static and runtime memory map of romable kernel image along with the sizes of various segments. The static map provides the addresses of various segments as they appear in ROM/disk image. The runtime memory map provides addresses of various segments as they would appear in kernel address space at run time.

SEE ALSO

Utility Programs - `mkimage, mkramdisk`

NAME

`ln` – make links between files

SYNOPSIS

ln [-s] *name1* **[** *name2***]**

ln [-s] *file ... directory*

DESCRIPTION

In the first form, `ln` creates a link called *name2* to existing file *name1*. In the second form, `ln` creates a link in *directory* for each of the *files*, using the base name of each file. If only one name is given, then the second form is assumed and directory is taken to be "**.**".

A " link" is a directory entry which refers to a file. A single file may have many links to it. Two kinds of links exist in LynxOS: "hard" links and "symbolic" links. A hard link to a file is a normal directory entry which refers to the inode of the file. There is no distinction between two hard links to the same file, other than the pathnames. Hard links may not span file systems, and they may not refer to directories (this is to prevent cycles in the directory structure). A file is actually removed only when it has no links; removing a link to a file with other links does not destroy the file. `ln` by default creates hard links.

A symbolic link to a file is a special file containing the pathname to which the link refers. The file referenced by the link file is used whenever any operation is performed on the link. The only way to obtain information about the link file itself is through the `readlink` or `lstat` system calls. Symbolic links may span file systems and may also refer to directories. The **-s** option instructs `ln` to create symbolic links.

SEE ALSO

Utility Programs - `rm, cp`

System Calls - `link(), readlink(), stat(), symlink(), unlink()`

NAME

login – log into LynxOS

SYNOPSIS

login [*username***] [-h** *hostname***] [-f** *username***] [***terminal_type***]**

DESCRIPTION

login begins a terminal session under a specified user ID. It is usually used when a user wishes to sign on the system, but it may be used anytime during a session to change from one user ID to another. If login is not given an argument, then it will prompt for a user name.

If login finds the requested user name in the /etc/passwd file it chains to a program (usually the shell) associated with that user name. The new program's environment includes values for the environment variables PATH, HOME, USER, SHELL, EXECSHELL, TERM, and CONFIG, describing the execution search path, the login directory name, the user name, the name of the new program, and the terminal type. The first argument (argv[0]) to the new program is constructed by prepending a dash to the program name.

Note that when login is started from the shell, a sub-process is started. When that process terminates (after login chains to the appropriate program) the original shell is resumed.

If the file /etc/motd exists, its contents are printed to the screen before the shell is started. It is possible for individual users to prevent the message of the day from being printed by creating a file called .hushlogin in their home directory. The contents of .hushlogin do not matter; its existence is all that is required.

OPTIONS

-h *hostname* — *hostname* is the name of the remote host where the login originated. login includes this information in the /etc/utmp and /etc/adm/wtmp files.

-f *username*	*username* is the name of the remote user where the login originated. `login` includes this information in the `/etc/utmp` and `/etc/adm/wtmp` files.

NOTE

The **-h** option is not for use by humans. It is used by `rlogind`, `telnetd`, and other servers so that the remote *hostname* can be included in the appropriate record keeping files.

The **-f** option is not for use by humans. It is used by `rlogind`, `telnetd`, and other servers so that the username is pre-verified for remote login.

FILES

`/etc/passwd`	User name information file.
`/etc/ttys`	`tty` information file.
`/etc/tconfig`	`tty` configuration table.
`/etc/motd`	Message of the day file.
`~/.hushlogin`	Prevents login from printing non-error messages.

SEE ALSO

Utility Programs - `passwd, dlsh`

NAME

`lpc` – line printer control program

SYNOPSIS

lpc [*command* **[** *argument* **...]]**

DESCRIPTION

`lpc` is used by the system administrator to control the operation of the line printer system. For each line printer configured in `/etc/printcap`, `lpc` may be used to:

1. disable or enable a printer.
2. disable or enable a printer's spooling queue.
3. rearrange the order of jobs in a spooling queue.
4. find the status of printers, and their associated spooling queues and printer daemons.

Without any arguments, `lpc` will prompt for commands from the standard input. If arguments are supplied, `lpc` interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing `lpc` to read commands from a file. Commands may be abbreviated: the following is a list of recognized commands.

`? [`*command...*`]`
`help [`*command...*`]`
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

`abort {all |` *printer...*`}`
: Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by `lpr`) for the specified printers.

`clean {all |` *printer...*`}`
: Remove any temporary files, data files, and control files that cannot be printed (i.e., do not form a complete printer job) from the specified printer queue(s) on the local machine.

`disable {all |` *printer...*`}`
: Turn the specified printer queues off. This prevents new printer jobs from entered into the queue by `lpr`.

`down {all | printer}` *message...*
: Turn the specified printer queue off, disable printing and put *message* in the printer status file. The message doesn't need to be quoted, the remaining arguments are treated like echo. This is normally used to take a printer down and let others know why (`lpq` will indicate the printer is down and print the status message).

`enable {all |` *printer...*`}`
: Enable spooling on the local queue for the listed printers. This will allow `lpr` to put new jobs in the spool queue.

`exit`
`quit`
: Exit from `lpc`.

`restart {all |` *printer...*`}`
: `lpq` will report that there is no daemon present when this condition occurs. If the user is the super-user, try to abort the current daemon first (i.e., kill and restart a stuck daemon).

`start {all |` *printer...*`}`
: Enable printing and start a spooling daemon for the listed printers.

`status {all |` *printer...*`}`
: Display the status of daemons and queues on the local machine.

`stop {all |` *printer...*`}`
: Stop a spooling daemon after the current job completes and disable printing.

`topq` *printer* `[`*jobnum...*`]` `[`*user ...*`]`
Place the jobs in the order listed at the top of the printer queue.

`up {all |` *printer ...*`}`
Enable everything and start a new printer daemon. Undoes the effects of `down`.

FILES

`/etc/printcap`	Printer description file.
`/usr/spool/*`	Spool directories.
`/usr/spool/*/lock`	Lock file for queue control.

DIAGNOSTICS

`?Ambiguous command`
Abbreviation matches more than one command.

`?Invalid command`
No match was found.

`?Privileged command`
Command can be executed by root only.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `lpd, lpr, lpq, lprm`

File Formats - `printcap`

NAME

`lpd` – line printer daemon

SYNOPSIS

`lpd [ -l]`

DESCRIPTION

`lpd` is the line printer daemon (spool area handler) and is normally invoked at boot time from the rc file. It makes a single pass through the `printcap` file to find out about the existing printers and prints any files left after a crash. It then uses the system calls `listen` and `accept` to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The **`-l`** flag causes `lpd` to log valid requests received from the network. This can be useful for debugging purposes.

Access control is provided by two means. First, all requests must come from one of the machines listed in the file `/etc/hosts.equiv` or `/etc/hosts.lpd`. Second, if the **`rs`** capability is specified in the printcap entry for the printer being accessed, `lpr` requests will only be honored for those users with accounts on the machine with the printer.

The file `minfree` in each spool directory contains the number of disk blocks to leave free so that the line printer queue won't completely fill the disk. The `minfree` file can be edited with your favorite text editor.

The file lock in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for `lpr`, `lpq`, and `lprm`. After the daemon has successfully set the lock, it scans the directory for files beginning with `cf`. Lines in each `cf` file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

`J` Job Name. String to be used for the job name on the burst page.

C	Classification. String to be used for the classification line on the burst page.
L	Literal. The line contains identification info from the password file and causes the banner page to be printed.
T	Title. String to be used as the title for `pr`.
H	Host Name. Name of the machine where `lpr` was invoked.
P	Person. Login name of the person who invoked `lpr`. This is used to verify ownership by `lprm`.
M	Send mail to the specified user when the current print job completes.
f	Formatted File. Name of a file to print that is already formatted.
l	Like **f** but passes control characters and does not make page breaks.
p	Name of a file to print using `pr` as a filter.
t	Troff File. The file contains `troff` output (cat phototypesetter commands).
n	Ditroff File. The file contains device-independent `troff` output.
d	DVI File. The file contains `Tex` output (DVI format from Stanford).
g	Graph File. The file contains data produced by `plot(3X)`.
c	`cifplot` File. The file contains data produced by `cifplot`.
v	The file contains a raster image.
r	The file contains text data with FORTRAN carriage control characters.
1	`troff` Font R. Name of the font file to use instead of the default.

2	`troff` Font I. Name of the font file to use instead of the default.
3	`troff` Font B. Name of the font file to use instead of the default.
4	`troff` Font S. Name of the font file to use instead of the default.
W	Width. Changes the page width (in characters) used by `pr` and the text filters.
I	Indent. The number of characters to indent the output by (in ascii).
U	Unlink. Name of file to remove upon completion of printing.
N	File Name. The name of the file which is being printed, or a blank for the standard input (when `lpr` is invoked in a pipeline).

If a file cannot be opened, a message will be logged via `syslog` using the `LOG_LPR` facility. `lpd` will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

`lpd` uses `flock` to provide exclusive access to the lock file and to prevent multiple daemons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control filename of the current job being printed. The second line is updated to reflect the current status of `lpd` for the programs `lpq` and `lprm`.

OPTIONS

-l	Causes `lpd` to log valid requests received from the network. This can be useful for debugging purposes.

FILES

`/etc/printcap`	Printer description file.
`/usr/spool/*`	Spool directories.

`/usr/spool/*/minfree`	Minimum free space to leave.
`/dev/lp*`	Line printer devices.
`/dev/printer`	Socket for local requests.
`/etc/hosts.equiv`	Lists machine names allowed printer access.
`/etc/hosts.lpd`	Lists machine names allowed printer access, but not under same administrative control.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs -	`lpc, pac, lpr, lpq, lprm`
System Calls -	`syslog()`
File Formats -	`printcap`
Documents -	`4.2BSD Line Printer Spooler Manual`

NAME

`lpq` – spool queue examination program

SYNOPSIS

lpq[*options* **] [***job# ...* **] [***user ...* **]**

DESCRIPTION

`lpq` examines the spooling area used by `lpd` for printing files on the line printer and reports the status of the specified jobs or all jobs associated with a user.

For each job submitted (each invocation of `lpr`), `lpq` reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number that the user may supply to `lprm` for removing a specific job), and the total size in bytes. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First In First Out). If you use `lpr` as a sink in a pipeline, `lpq` cannot determine the filenames in the job. In this case, `lpq` labels the file name as "standard input".

Without any arguments, `lpq` reports on any jobs currently in the queue.

If `lpq` warns that there is no daemon present (i.e. due to some malfunction), you can use the `lpc` command to restart the printer daemon.

OPTIONS

+	Displays the spool queue until it empties.
+ *n*	Displays the spool queue until it empties, but sleeps *n* seconds in between scans of the queue.
-l	Prints information about each of the files comprising the job. Normally, only as much information as will fit on one line is displayed.
-P *printer*	Specifies a particular printer. Without this option, `lpq` checks the default line printer or the value of the `PRINTER` variable in the environment.

All other arguments supplied are interpreted as user names or job numbers to filter out only those jobs of interest.

FILES

`/etc/termcap`	For manipulating the screen for repeated display.
`/etc/printcap`	To determine printer characteristics.
`/usr/spool/*`	The spooling directory, as determined from printcap.
`/usr/spool/*/cf*`	Control files specifying jobs.
`/usr/spool/*/lock`	The lock file to obtain the currently active job.

DIAGNOSTICS

Unable to open various files.

The lock file being malformed.

Garbage files when there is no daemon active, but files in the spooling directory.

BUGS

Due to the dynamic nature of the information in the spooling directory `lpq` may report unreliably. Output formatting is sensitive to the line length of the terminal.

Sometimes this results in widely-spaced columns.

SEE ALSO

Utility Programs - `lpr, lprm, lpc, lpd`

NAME

`lpr` – off line print

SYNOPSIS

lpr [*options* **] [** *filename ...* **]**

DESCRIPTION

`lpr` uses a spooling daemon to print the named files when facilities become available. If no names appear, `stdin` is assumed.

The single letter options below are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

OPTIONS

– #*num* Prints out num copies of each file named. For example,

```
lpr -#3 foo.c bar.c more.c
```

would result in three copies of the file `foo.c`, followed by three copies of the file `bar.c`, etc.

On the other hand,

```
cat foo.c bar.c more.c | lpr -#3
```

will give three copies of the concatenation of the files.

– *num font* Specifies a font to be mounted on *font* position **i**. *num* is a font number from 1 to 4. The daemon will construct a `.railmag` file referencing:

```
/usr/lib/vfont/name.size
```

–c Assumes the files contain data produced by `cifplot`.

–C Takes the following argument as a job classification for use on the burst page. For example,

```
lpr -C EECS foo.c
```

	causes the system name (the name returned by hostname) to be replaced on the burst page by EECS, and the file `foo.c` to be printed.
-d	Assumes the files to contain data from `tex` (DVI format from Stanford).
-f	Uses a filter which interprets the first character of each line as a standard FORTRAN carriage control character.
-g	Assumes the files to contain standard plot data as produced by the `plot(3X)` routines (see also `plot(1G)` for the filters used by the print spooler).
-h	Suppresses the printing of the burst page.
-i *num*	Causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.
-J *job*	Takes *job* as the job name to print on the burst page. Normally, the the first file's name is used.
-l	Uses a filter which allows control characters to be printed and suppresses page breaks.
-n	Assumes the files to contain data from `ditroff` (device independent `troff`).
-p	Uses `pr` to format the files (equivalent to *print*).
-P *printer*	Forces output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable `PRINTER` is used.
-r	Removes the file upon completion of spooling or upon completion of printing (with the **-s** option).
-m	Sends mail when printing is finished.
-s	Uses symbolic links to copy files to the spool directory. This option uses `symlink` to link data files rather that trying to copy them so large files can be printed. This means the files should not be modified or removed until they have been printed.

-t	Assumes the files to contain data from `troff` (cat phototypesetter commands).
-T *title*	Takes the *title* for the title used by `pr` instead of filename.
-v	Assumes the files to contain a raster image for devices like the Benson Varian.
-w *num*	Takes *num* to be the page width for `pr`.

FILES

`/etc/passwd`	Personal identification.
`/etc/printcap`	Printer capabilities database.
`/usr/lib/lpd*`	Line printer daemons.
`/usr/spool/*`	Directories used for spooling.
`/usr/spool/*/cf*`	Daemon control files.
`/usr/spool/*/df*`	Data files specified in `cf` files.
`/usr/spool/*/tf*`	Temporary copies of `cf` files.

DIAGNOSTICS

If you try to spool too large a file, it will be truncated. `lpr` will object to printing binary files. If a user other than root prints a file and spooling is disabled, `lpr` will print a message saying so and will not put jobs in the queue. If a connection to `lpd` on the local machine cannot be made, `lpr` will say that the daemon cannot be started. Diagnostics may be printed in the daemons log file regarding missing spool files by `lpd`.

BUGS

Fonts for `troff` and `tex` reside on the host with the printer. It is currently not possible to use local font libraries.

ACKNOWLEDGEMENTS

This software was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs -	`lpq, lprm, lpc, lpd`
System Calls -	`symlink()`
File Formats -	`printcap`

NAME

lprm – remove jobs from the line printer spooling queue

SYNOPSIS

lprm [*options* **] [** *job#* **] [** *user(s)* **]**

DESCRIPTION

lprm removes a job or jobs from a printer's spool queue. Since the spooling directory is protected from users, using lprm is normally the only method by which a user may remove a job.

Without any arguments, lprm deletes the currently active job if it is owned by the user who invoked lprm.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the lpq program. For example:

```
%lpq -l
1st:ken [job #013ucbarpa]
 (standard input) 100 bytes
%lprm 13
```

lprm announces the names of any files it removes and is silent if there are no jobs in the queue that match the request list.

lprm kills off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, lprm automatically starts a new one upon completion of file removals.

OPTIONS

– Causes lprm to remove all jobs that a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and hostname on the machine where the lpr command was invoked.

-P *printer*	Specifies the queue associated with a specific printer (otherwise the default printer, or the value of the `PRINTER` variable in the environment is used).
job#	Dequeues an individual job by specifying its job number.
user(s)	Causes `lprm` to remove any jobs belonging to the specified user or users. Only the super-user can use this option of `lprm`.

FILES

`/etc/printcap`	Printer characteristics file.
`/usr/spool/*`	Spooling directories.
`/usr/spool/*/lock`	Lock file used to obtain the pid of the current daemon and the job number of the currently active job.

DIAGNOSTICS

"Permission denied" if the user tries to remove files other than his own.

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `lpr, lpq, lpd`

NAME

`lptest` – generate line printer ripple pattern

SYNOPSIS

`lptest [` *length* **`[`** *count* **`] ]`**

DESCRIPTION

`lptest` writes the traditional "ripple test" pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the default length of 79 is inappropriate.

The *count* argument is the number of output lines to be generated if the default count of 200 is inappropriate. Note that if you specify *count*, you must also specify *length*.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

NAME

`ls` – show directory contents and file information

SYNOPSIS

ls [-1ACFLRacdfgilnqrstu] [-T *types* **]** *file ...*

DESCRIPTION

`ls` writes to standard output information requested by options for each *file* given. If a *file* is a directory, then the entire contents of that directory are written. Output is sorted alphabetically by default, but can optionally be sorted by access or modification time. If no *file* is given, the current directory is assumed.

The list of *file*s is sorted alphabetically; non-directory arguments are always processed before directory arguments.

OPTIONS

-1	One line of output is generated for each file. This option is assumed when standard output is not to a terminal.
-A	When listing directory contents, all files are included, including those whose names begin with "`.`" but excluding "`.`" and "`..`".
-C	`ls` will list as many files per output line as possible. This option is assumed when standard output is a terminal.
-F	`ls` appends markers to file names in the listing to indicate file type, as follows:
/	Directory.
&	Named pipe (FIFO).
=	Socket (not implemented in LynxOS).
@	Symbolic link.
*****	Executable regular file.
|	IPC special file

–L	For each argument or directory element which is a symbolic link, information about the file or directory referenced by the link (instead of the link itself) is used for printing, sorting, and / or marking the output.
–R	Each subdirectory encountered while scanning a directory is itself examined.
–T *types*	Only files whose type is one of those specified by *types* will be shown. The *types* list should be a string of one or more of **–**, **b**, **c**, **d**, **f**, **l**, **p**, **r**, **s**, **i**, **I**, and **+**, as described below.
–a	When listing directory contents, all files are printed, including files which begin with "**.**".
–c	The time value examined for printing (**–l**) and sorting (**–u**) will be the inode change time instead of the modification time.
–d	Instructs `ls` to give information about directory arguments themselves instead of listing the contents of the directories.
–f	Each argument is interpreted as a directory, and contents are listed in the order encountered (no sort is performed). The **–f** option disables **–l**, **–t**, **–s**, and **–r**, and enables **–a**.
–g	Group ID of each file is printed (only effective when **–l** is also given).
–i	The inode number of each file is printed.
–l	Extended information is given for each file, including file mode, number of links, owner, size in bytes, and time (usually modification time, but optionally `access` or `inode` change time). If a file is a special file, the major and minor numbers are given in place of file size. If a file is a symbolic link, then the target of the link is given after the file name.
–n	Restricts listing to files created on the current system date.

-q	Forces non-printable characters in file names to be printed as "**?**". This option is assumed when standard output is a terminal.
-r	Reverses sort order.
-s	Lists size of each file, in 512-byte blocks.
-t	Sorts by time rather than by name.
-u	Uses time of last access rather than modification time for printing (**-l**) and sorting (**-t**).

Mode Information

The file mode given with the **ls -l** option is printed as an ten character string. The first character gives general file type:

-	Regular file.
f	Regular file.
r	Regular file.
b	Block special file.
c	Character special file.
d	Directory file.
p	Named pipe (FIFO) symbolic link.
s	Socket.
+	Contiguous file.
i	Non persistent IPC special file.
I	Persistent IPC special file.

The next nine characters tell whether or not read, write, and execute permissions are on for the owning user, owning group, and others:

r	Read permission.
w	Write permission.

x Execute permission.

The permissions are printed in the order read-write-execute; a dash in a position means that permission is off. For directory files, execute permission is interpreted as permission to search the directory. If the “sticky bit” in the mode is turned on (see the system call `chmod`), the last character of the mode string is printed as **t** rather than the usual **x** or **-**.

SEE ALSO

Utility Programs -	`chmod, chown, ln, wc`
System Calls -	`chmod(), chown(), stat(), utimes()`
Library Functions -	`directory, getpwent(), scandir()`

NAME

`l-strip` – legacy Lynx utility for removing symbol information

SYNOPSIS

`l-strip [-lxbrV]` *file ...*

DESCRIPTION

`l-strip` is a legacy Lynx utility. We recommend that you use the GNU tools version where possible. See `strip(1)` for more information.

`l-strip` removes the symbol table and relocation information from each object or executable *file*. This is useful in reducing the file storage overhead of a program that has been fully tested.

`l-strip` saves the original version of each *file* in *file*`.unstripped`.

OPTIONS

`-l` Strip line number information and not symbol table information.

`-x` Do not strip static or external symbol information.

`-b` Same as **`-x`** option, but also do not strip scope information (e.g. beginning and end of block delimiters).

`-r` Do not strip static or external symbol information, or relocation information.

`-V` Print the version of this `l-strip` command on standard output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, `l-strip` will complain and terminate without stripping the *file* unless **`-r`** option is used.

The **`-l`**, **`-x`**, **`-b`** and **`-r`** options cannot be performed on `a.out` format files.

FILES

`./striptemp`	a temporary file

SEE ALSO

Utility Programs - `l-ld`, `l-cc`

NAME

`l-tar` – legacy Lynx utility to combine files into an archive

SYNOPSIS

l-tar {crtux} [BIRXbfhklmopvw] [*blksize***] [***tape***] [-C dir]**
[-P *prefix***]** *files...*

DESCRIPTION

The `l-tar` utility is a legacy Lynx version of `tar`. It is provided for backwards-compatibility with previous release of LynxOS. However, we strongly recommend that you use the GNU version of `tar` (formerly called `gnutar`) when possible. The `l-tar` utility should not be used on files created with GNU `tar`.

`l-tar` is historically intended to combine files into an archive on a tape. The archive need not actually be a tape. Each *file* is copied to or from the archive; directory files are taken to mean the entire subtree beneath the directory.

When creating an archive, `l-tar` provides two special mechanisms to allow the archive to reflect a different directory structure than the source of the archived files. A pair of arguments **-C** *dir* causes the `l-tar` process to switch to directory *dir* before interpreting subsequent file names. A pair **-P** *prefix* causes the string *prefix* to be prepended to file names as they are recorded in the `l-tar` output file. Any number of directory-switch and prefixing commands can be given in the file list. Note that once the `l-tar` process has changed its working directory, subsequent relative pathnames given in **-C** requests will be interpreted relative to the then current directory.

OPTIONS

Commands

The `l-tar` command string must be the first argument given. It need not (but can) begin with a dash. `l-tar` basic operation is dictated by one of the following letters embedded in the string:

c Create the output archive anew.

r Append files to the end of an existing archive.

t Pertinent information about files on an existing archive is printed to the standard output. If no *file* arguments are given to select files in the archive, all files in the archive are shown.

u Each *file* is appended to the archive only if it is not there already, or if it has been changed since last put there. Note that the older versions of files are not deleted.

x Each *file* is extracted from the tape. If a *file* is actually a directory, the entire subtree is extracted. Shell wildcard operators are recognized within *file* arguments.

Modifiers

The following characters may also appear in the command string, and alter the behavior of the basic `l-tar` commands.

B This command has no effect and exists for backward compatibility only. Older versions of `l-tar` truncate the final block of an archive so that its blocking factor is not always the same as the other blocks. The **B** command would force `l-tar` to pad the final block with null records up to the blocking factor. The current version of `l-tar` always pads the final block, regardless of the **B** command.

I Ignore `checksum`. Use if `checksum` error is reported and you would like to extract the files anyway.

R Suppresses attempts to rename existing files to resolve type conflicts with the archive (see below).

X Instructs `l-tar` to use POSIX-format `l-tar` headers, instead of (simpler and less powerful) UNIX format headers. LynxOS `l-tar` can read either format.

b Instructs `l-tar` to recognize the argument following the key as the blocking factor. `l-tar` constructs the archive as groups of 512-byte records. The default blocking factor is 20 records; the maximum blocking factor is 256 records. Each block is written with one `write` request, and so will constitute a physical record on a raw tape device. The block size is determined automatically when reading an archive.

Note: If you use both the **b** and the **f** modifiers, be careful to enter their arguments in the correct order. To illustrate consider these command lines:

```
l-tar cbf 20 archive
l-tar cfb archive 20
```

f Instructs `l-tar` to recognize the argument following the key as the name of the archive. If omitted, `l-tar` writes to the file `/dev/l-tar.default`. If the file name given is a single dash ('–'), `l-tar` writes to standard output or reads from standard input. `l-tar` is thus the method of choice for moving directory subtrees:

```
cd source; l-tar cf - | -( cd dest; l-tar xf - )-
```

h Instructs `l-tar` to follow symbolic links when creating an archive. Normally, symbolic links are recorded as such in the output.

k Instructs `l-tar` not to repair access times of files read during archive creation. Normally `l-tar` ensures that access and modification times of files read during archive creation are not affected by the process.

l Causes `l-tar` to produce error messages during an extraction when attempts to re-create hard links fail.

m Prevents restoration of modification times. The time stamp on extracted files will be equal to the time at the inception of the `l-tar` process.

o Prevents `l-tar` from explicitly recording directory files in the archive.

p Causes present `umask` information to be ignored when restoring modes of extracted files.

v Sets verbose mode, so that `l-tar` prints information about each file processed. When checking the contents of an archive (the **t** command), the verbose option gives more information about the files. Normally, `l-tar` operates silently.

w Causes `l-tar` to request verification of each transaction.

CONFLICTS DURING EXTRACTION

When `l-tar` is extracting files from an archive into a file system, it can encounter conflicts in file type between existing files and files in the archive. In such cases, `l-tar` will attempt to rename the existing file and then retry the extraction. Files are renamed by prepending a dot and appending a suffix of the form:

```
,l-tar-03-Sep-1989-10:42:19
```

In other words, the suffix includes the tag `,l-tar-` followed by the date and time of the extraction. All files renamed in a singled run of `l-tar` will have the same date tag. Because the new file names begin with a dot, they will not appear in directory listings. Care should be taken that old renamed files do not clutter the disk. Conflicts arise when one of the following is true:

- The file in the archive is a directory, and the existing file is not.
- The file in the archive is a hard or symbolic link, and the existing file is simply present.
- The file in the archive is a regular file, and the existing file is not.
- The file in the archive is a regular file, and the existing file is an executing program.

If the **R** flag is present in the `l-tar` command string, renames will not be attempted, and messages to that effect are printed should conflicts arise.

EXAMPLES

To collect all files in a subdirectory called `dir` into an archive called `archive`:

l-tar cf archive dir

To print the contents of the `archive`:

l-tar tvf archive

To extract the files from the archive and create a duplicate of `dir` in the subdirectory `backup`:

cd backup
l-tar xf ../archive

To copy the contents of one directory to another (using dlsh):

cd *fromdir*
l-tar cf - | -(cd *destdir*; **l-tar xf -)-**

or:

cd *fromdir*
l-tar cf - -P *destdir* **. | l-tar xf -**

To copy the root file system hierarchy to another file system mounted under `/mnt` (using `dlsh`):

cd /
l-tar cf - $drop(mnt, **) | -(cd /mnt; l-tar xf -)-

or:

cd /
l-tar cf - -P /mnt $drop(mnt, **) | l-tar xf -

To copy files from a local machine to a directory on a remote machine machine (using `dlsh`):

l-tar cf - *files* **| rsh** *machine* **"-(cd** *dir*; **l-tar xf -)-"**

To extract all files on `/dev/rtape`:

```
l-tar xvf /dev/rtape `./fred/jane/*'
```

NOTE

Because some streaming tape drives do not provide support for backwards seeks, the **r** and **u** flags may not work. Also, some tape drives do not allow blocks of arbitrary size to be written. For such devices, the `l-tar` block size will not affect the physical arrangement of data on the tape and should be chosen to be large to increase performance.

The `l-tar` utility should not be used on files created with GNU `tar`.

SEE ALSO

Utility Programs - `libr`

NAME

`lynxos_agent` – SNMP LynxOS Sub Agent Daemon

SYNOPSIS

`/net/lynxos_agent`

DESCRIPTION

The `lynxos_agent` supports key LynxOS kernel parameters as a Simple Network Management Protocol (SNMP) sub-agent. This subagent is invoked from the `/etc/config/netservice.snmp` startup script. The SNMP installation script configures the `/etc/sysconfig` file to automatically begin the `lynxos_agent`.

The following group support is still evolving and the following groups of Management Information Base (MIB) are currently supported:

```
iso(1).org(3).dod(6).internet(1).private(3).enterprises(1).Lynx
RTS(1685).lynxos(1).system(1)
```

```
iso(1).org(3).dod(6).internet(1).private(3).enterprises(1).Lynx
RTS(1685).lynxos(1).file_system(2)
```

The following read-only objects are supported under the system subgroup.

```
Size of the Process table : system.1  ( integer )
Size of the Driver table  : system.2  ( integer )
Number of (4K) FreePages  : system.3  ( integer )
Number of incode inodes   : system.4  ( integer )
Max. no. of Open files    : system.5  ( integer ) per process
Size of the file table    : system.6  ( integer )
Size of the User Sem.Table: system.7  ( integer )
Size of the Shared memory : system.8  ( integer ) table
Size of the Char. Device  : system.9  ( integer ) table
Size of the Block Device  : system.10 ( integer ) table
Size of the Stream Task   : system.11 ( integer ) table
Size of the System V      : system.12 ( integer ) message table
Size of the System V      : system.13 ( integer ) semaphore table
Size of the System V      : system.14 ( integer ) shared mem table
Max. no. of access groups : system.15 ( integer ) per process
Size of a pipe in the     : system.16 ( integer ) system
Max. no. of mounted file  : system.17  ( integer ) system
Max. no. of threads in the: system.18  ( integer ) system
No. of free virtual (4K)  : system.19  ( integer ) pages in system
Status of the virtual     : system.20  ( integer ) memory system
System Name               : system.21  ( string )
HostName                  : system.22  ( string )
Release of LynxOS         : system.23  ( string )
Version under this release: system.24  ( string ) of LynxOS
The hardware on which     : system.25  ( string ) LynxOS is running
```

The mounted-file-system table information is provided under the file_system sub-group and contains the following information (as `file_system(2).1.1` sub-group; all read-only components).

```
DeviceName          (string)
Pathname            (string)
mount_type          (string)
blksize             (integer)
No.of blocks        (integer)
No.of used blocks   (integer)
No.of free blocks   (integer)
No.of inodes        (integer)
No.of used inodes   (integer)
No.of free inodes   (integer)
```

SEE ALSO

Utility Programs - `snmpd`, `mibIIagent`

File Formats - `config.snmp()`

NAME

`m4` – general-purpose macro processor

SYNOPSIS

m4 [-D *name***=***val* ... **] [-U** *name* ... **]**

DESCRIPTION

`m4` is a macro processor intended as a front end for language translators that do not have built-in macro processing capability. `m4` reads `stdin`, and the processed text is written on `stdout`.

OPTIONS

-D *name*=*val* Defines *name* to *val* or to null if *val* is absent.

-U *name* Overrides the first definition of *name* in the source file.

Macro calls have the form:

```
name(arg1, arg2, ... , argn)
```

The initial parenthesis must immediately follow the name of the macro. If the name of a defined macro is not followed by a parenthesis it is taken to be a call of that macro with no arguments, as in `name()`.

Macro names may include alphabetic letters and digits. Leading unquoted blanks, tabs and newlines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching close parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. (This is typically referred to as “inside-out macro expansion” in the macro expansion community.) After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

`m4` makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

`define`
: syntax: `define(`*name* `[ ,` *val* `] )`
The second argument is installed as the value of the macro whose name is the first argument. If there is no second argument, the value is null. Each occurrence of `$`*n* in the replacement text, where *n* is a digit, is replaced by the *n*'th argument. Argument `0` is the name of the macro; missing arguments are replaced by the null string.

`defn`
: syntax: `defn`*(name [, name, ...])*
Returns the quoted definition of its argument(s). Useful in renaming macros.

`undefine`
: syntax: `undefine(`*name* `[ ,` *name*`, ... ] )`
Removes the definition of the named macro(s). If there is more than one definition for the named macro, (due to previous use of `pushdef`) all definitions are removed.

`pushdef`
: syntax: `pushdef(`*name* `[ ,` *val* `] )`
Like define, but saves any previous definition by stacking the current definition.

`popdef`
: syntax: `popdef(`*name* `[ ,` *name*`, ... ] )`
Removes current definition of its argument(s), exposing the previous one if any.

`ifdef`
: syntax: `ifdef(`*name*`,is-defined [ ,not-defined ])`
If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null.

`shift`
: syntax: `shift(`*arg*`,` *arg*`, ... )`
Returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

`changequote`	syntax: `changequote(lqchar, rqchar)` Change quote symbols to the first and second arguments. With no arguments, the quotes are reset back to the default characters “ and ”.
`changecom`	syntax: `changecom(lcchar, rcchar)` Change left and right comment markers from the default `#` and `newline`. With no arguments, the comment mechanism is reset back to the default characters. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected.
`divert`	syntax: `divert(`*divnum*`)` `m4` maintains 10 output streams, numbered 0-9. Initially stream 0 is the current stream. The divert macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 disappears.
`undivert`	syntax: `undivert(`*divnum* `[ ,` *divnum*`, ... ] )` Causes immediate output of text from diversions named as argument(s), or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text. At the end of input processing, `m4` forces an automatic `undivert`, unless `m4wrap` is defined.
`divnum`	syntax: *divnum*`()` Returns the value of the current output stream.
`dnl`	syntax: `dnl()` Reads and discards characters up to and including the next newline.
`ifelse`	syntax: `ifelse(`*arg1*`,`*arg2*`,if-same [,` *arg*`, ... ] )` `Ifelse` must have three or more arguments. If the *arg1* is the same string as *arg2*, then the value of `ifelse` is the third argument, `if-same`. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

`incr` syntax: `incr(`*num*`)`
Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

`decr` syntax: `decr(`*num*`)`
Returns the value of its argument decremented by 1.

`eval` syntax: `eval(`*expression*`)`
Evaluates its argument as a constant expression, using integer arithmetic. The evaluation mechanism is very similar to that of the standard C preprocessor. The expression can involve only integer constants and character constants, possibly connected by the binary operators

```
* / % + >> << < > <= >= == !=& ^ | && ||
```

or by the unary operators

```
- ~ !
```

or by the ternary operator `? :`.

Parentheses may be used for grouping. Octal numbers may be specified as in C.

`len` syntax: `len(`*string*`)`
Returns the number of characters in its argument.

`index` syntax: `index(`*search-string*`,` *string*`)`
Returns the position in its first argument where the second argument begins (zero origin), or `-1` if the second argument does not occur.

`substr` syntax: `substr(`*string*`,` *index* `[ ,` *length* `] )`
Returns a substring of its first argument. The second argument is a zero origin number selecting the first character (internally treated as an expression); the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

`translit`	syntax: `translit(`*source*`, from [ , to ] )` Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. If the third argument is shorter than the second, all extra characters in the second argument are deleted from the first argument. If the third argument is missing altogether, all characters in the second argument are deleted from the first argument.
`include`	syntax: `include(`*filename*`)` Returns the contents of the file named in the argument.
`sinclude`	syntax: `sinclude(`*filename*`)` Is identical to include except that it says nothing if the file is inaccessible.
`paste`	syntax: `paste(`*filename*`)` Returns the contents of the file named in the argument without any processing, unlike include.
`spaste`	syntax: `spaste(`*filename*`)` Is identical to paste, except that it says nothing if the file is inaccessible.
`syscmd`	syntax: `syscmd(`*command*`)` Executes the program given in the first argument. The shell is used to interpret *command*, so normal shell features are available. No value is returned.
`sysval`	syntax: `sysval()` Is the return code from the last call to `syscmd`.
`maketemp`	syntax: `maketemp(`*string*`)` Fills in a string of six `X`s (like `XXXXXX`) in its argument with the current process ID.
`m4exit`	syntax: `m4exit([` *exitcode* `])` Causes immediate exit from `m4`. Argument 1, if given, is the exit code; the default is 0.
`m4wrap`	syntax: `m4wrap(`*arg*`)` The argument will be pushed back when `m4` reaches end of input and is preparing to exit.

`errprint` syntax: `errprint(`*str* `[ ,`*str*`, ... ])`
Prints its argument(s) on `stderr`. If there is more than one argument, each argument is separated by a space in the output.

`dumpdef` syntax: `dumpdef([ `*name*`, `*name*`, ... ])`
Prints current names and definitions, for the named items, or for all defined names if no arguments are given.

EXAMPLE

Example 1. Here is a simple `m4` program to get you started. What we wanted to do was use one input document to generate two different documents after `m4` processing. The content of the two documents is the same except for a couple of lines. We wanted to avoid having to edit two separate similar documents.

To run the program, type:

```
m4 -D customer_only=1 < m4_program
```

or

```
m4 -D internal_only=1 < m4_program
```

You can, of course, redirect the output to a file.

```
dnl # The m4 program begins here:
dnl # On the command line, you can specify the defines:
dnl # m4 -D customer_only=1 < m4_program > output_file
dnl # m4 -D internal_only=1 < m4_program > output_file
dnl # Or change the conditional print defines here (you will need
to switch
dnl # these depending on which document you want to print (see
below)
dnl define("customer_only", 1) dnl
dnl define("internal_only", 1) dnl
changequote(",") dnl
ifdef("internal_only","
```

Here is internal only text.

```
“,”
```

Here is customer only text.

```
“) dnl # end if then else internal only
```

Here is the part where the common text is.

```
ifdef(“internal_only”,”
```

Now we have internal only text.

```
“) dnl #end if then else
ifdef(“customer_only”,”
```

Now we have customer only text.

```
“) dnl #end customer_only
dnl # The m4 program ends here.
```

Example 2. The following program is fairly complicated. We want to create output of the form:

```
struct prehash {
  char *keyword;
  int hashval;
} keytab[] = {
  “foo”, 27,
  “bar”, 12,
  “baz”, 20,
  “”, 0
};
```

Given the input:

```
TSTART
  KEYWORD("foo")
  KEYWORD("bar")
  KEYWORD("baz")
TEND
```

(this input is at the end of the m4 program)

The components of the program are:

1. Replace the quote character with <>, so that <=open quote and >=close quote.
2. Replace the word TSTART with the string:
   ```
   struct prehash {<cr><tab>char \
   *keyword;<nl><tab>int hashval;<nl>} keytab[] = {
   ```
3. Replace the word TEND with the string:
   ```
   <tab>"", 0<nl>};
   ```
4. Define HASHSIZE to be the number 99.
5. For each input keyword ("foo", "bar", "baz") echo the word and then echo the calculated hash value of the word (the value of "b" + the value of "a" + the value of "r" mod HASHSIZE) where HASHSIZE is defined as 99.

In order to do the hash calculation, we use a recursive call to str, for the KEYWORD(bar). This is how the recursion works:

The ACSII value of 'b' is 98, 'a' is 97, 'r' is 114.

```
eval(str(bar",0)%HASHSIZE)
recurse: str(ar",eval($2 + 'b')) (on entry $2 is 0)
recurse: str(r",eval($2 + 'a'))   (on entry $2 is 0 + 'b' = 98)
recurse: str(",eval($2 + 'r'))  (on entry $2 is 98 + 'a'=195)
```

the ifelse sees that we have reached " (quote) and returns

```
$2 (195 + 'r' = 309)
```

The initial line becomes:

```
eval(309 % 99) = 12
```

To run the program, type:

```
m4 < m4_program
```

The `m4` program follows:

```
dnl # The m4 program begins here:
changequote(<,>)dnl
define(TSTART,
<struct prehash {
  char *keyword;
  int hashval;
} keytab[] = {>)dnl
define(TEND,<  ““, 0
};>)dnl
define(HASHSIZE,99)dnl
define(str,
  <ifelse($1,”,$2,
   <str(substr($1,1),<eval($2+’substr($1,0,1)’)>)>)
  >)dnl
dnl # Note that the quotes used in the preceding line are
<forward_quotes>.
define(hash,<eval(str(substr($1,1),0)%HASHSIZE)>)dnl
define(KEYWORD,<$1, hash($1),>)dnl
TSTART
  KEYWORD(“foo”)
  KEYWORD(“bar”)
  KEYWORD(“baz”)
TEND
dnl # The m4 program ends here.
```

Note that the Lynx `m4` allows a character to be added to a number. GNU `m4` does not. The following changes will allow the program to work under either Lynx `m4` or GNU `m4`:

Add:

```
define(charlist,<< !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
>>)dnl
```

(The above line must be all on one line; it has been spilt to be presented in this document.)

Replace:

```
<str(substr($1,1),<eval($2+'substr($1,0,1)')>)>)
```

With:

```
<str(substr($1,1),<eval($2+index(charlist,substr($1,0,1))+32)>)
>)
```

(32 is the start of the printable ASCII characters in the ASCII table.)

NOTE

`m4` was written by Mr. Ozan S. Yigit of MetaWare. He graciously placed the program in the public domain, and as such all source code for it (including modifications due to Lynx Real-Time Systems, should any come into existence) is available for a nominal handling fee. This does not imply any liability on the part of Mr. Yigit or Metaware; Lynx Real-Time Systems will support this software but will always maintain its free status.

SEE ALSO

Utility Programs - `cc`

NAME

`make` – GNU make utility to maintain groups of programs

SYNOPSIS

make [-f *makefile*] [*option*]... *target*...

DESCRIPTION

The purpose of the `make` utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them. This manual describes the GNU implementation of `make`, which was written by Richard Stallman and Roland McGrath. Our examples show C programs, since they are most common, but you can use `make` with any programming language whose compiler can be run with a shell command. In fact, `make` is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change.

To prepare to use `make`, you must write a file called the `makefile` that describes the relationships among files in your program, and the states the commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files.

Once a suitable makefile exists, each time you change some source files, this simple shell command:

```
make
```

suffices to perform all necessary recompilations. The `make` program uses the makefile data base and the last-modification times of the files to decide which of the files need to be updated. For each of those files, it issues the commands recorded in the data base.

`make` executes commands in the `makefile` to update one or more target *names*, where *name* is typically a program. If no **-f** option is present, `make` will look for the makefiles `GNUmakefile`, `makefile`, and `Makefile`, in that order.

Normally you should call your makefile either `makefile` or `Makefile`. (We recommend `Makefile` because it appears prominently near the beginning of a directory listing, right near other important files such as `README`.) The first name checked, `GNUmakefile`, is not recommended for most makefiles. You should use this name if you have a makefile that is specific to GNU `make`, and will not be understood by other versions of `make`. If `makefile` is '**–**', the standard input is read.

`make` updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

OPTIONS

–b
–m These options are ignored for compatibility with other versions of `make`.

–C *dir* Change to directory *dir* before reading the makefiles or doing anything else. If multiple **–C** options are specified, each is interpreted relative to the previous one: **–C / –C** etc is equivalent to **–C** /etc. This is typically used with recursive invocations of `make`.

–d Print debugging information in addition to normal processing. The debugging information says which files are being considered for remaking, which file-times are being compared and with what results, which files actually need to be remade, which implicit rules are considered and which are applied; everything interesting about how `make` decides what to do.

–e Give variables taken from the environment precedence over variables from makefiles.

–f *file* Use *file* as a makefile.

–i Ignore all errors in commands executed to remake files.

–I *dir* Specifies a directory *dir* to search for included makefiles. If several **–I** options are used to specify several directories, the directories are searched in the order specified. Unlike the arguments to other flags of `make`, directories given with **–I** flags may come directly after the flag: **-I***dir* is allowed, as well as **-I** *dir*. This syntax is allowed for compatibility with the C preprocessor's **-I** flag.

-j *jobs*
: Specifies the number of jobs (commands) to run simultaneously. If there is more than one **-j** option, the last one is effective. If the **-j** option is given without an argument, `make` will not limit the number of jobs that can run simultaneously.

-k
: Continue as much as possible after an error. While the target that failed, and those that depend on it, cannot be remade, the other dependencies of these targets can be processed all the same.

-l
-l *load*
: Specifies that no new jobs (commands) should be started if there are others jobs running and the load average is at least *load* (a floating-point number). With no argument, removes a previous load limit.

-n
: Print the commands that would be executed, but do not execute them.

-o *file*
: Do not remake the file *file* even if it is older than its dependencies, and do not remake anything on account of changes in *file*. Essentially the file is treated as very old and its rules are ignored.

-p
: Print the data base (rules and variable values) that results from reading the makefiles; then execute as usual or as otherwise specified. This also prints the version information given by the **-v** switch (see below). To print the data base without trying to remake any files, use

```
make -p -f/dev/null
```

-q
: *Question* mode. Do not run any commands, or print anything; just return an exit status that is zero if the specified targets are already up to date, nonzero otherwise.

-r
: Eliminate use of the built-in implicit rules. Also clear out the default list of suffixes for suffix rules.

-s Silent operation; do not print the commands as they are executed.

-S Cancel the effect of the **-k** option. This is never necessary except in a recursive `make` where **-k** might be inherited from the top-level `make` via `MAKEFLAGS` or if you set **-k** in `MAKEFLAGS` in your environment.

-t Touch files (mark them up to date without really changing them) instead of running their commands. This is used to pretend that the commands were done, in order to fool future invocations of `make`.

-v Print the version of the `make` program plus a copyright, a list of authors and a notice that there is no warranty. After this information is printed, processing continues normally. To get this information without doing anything else, use

```
make -v -f/dev/null
```

-w Print a message containing the working directory before and after other processing. This may be useful for tracking down errors from complicated nests of recursive `make` commands.

-W *file* Pretend that the target *file* has just been modified. When used with the **-n** flag, this shows you what would happen if you were to modify that file. Without **-n**, it is almost the same as running a `touch` command on the given file before running `make`, except that the modification time is changed only in the imagination of `make`.

FILES

`/usr/local/doc/gnumake.dvi` *The GNU Make Manual*

BUGS

See the chapter ‘Problems and Bugs’ in *The GNU Make Manual .*

AUTHOR

This manual page contributed by Dennis Morse of Stanford University. It has been reworked by Roland McGrath.

NAME

makeboot – install default bootstrap program

SYNOPSIS

makeboot [-b *device***] [-d** *device***] [-r** *device***]** *file*

DESCRIPTION

makeboot performs platform-specific operations to make a file bootable. It is typically used to make preboot, the kernel loader utility, bootable (see preboot(1)). makeboot can also be used to make a LynxOS kernel image directly bootable and, on PowerPC, 68k, and SPARC, it can be used to make a CD-ROM containing an ISO9660 file system bootable.

OPTIONS

-b *device* — Sets the boot device name (not supported on 68k). The argument *device* is where the LynxOS kernel is located. *device* may be given as a major-minor pair (*maj*, *min*) or as the name of a block device file. The boot device name can be different from the root file system device. The boot device name is used by preboot as the default boot device.

-d *tape_device* — PowerPC and 68k only. When this option is specified, makeboot will put the LynxOS kernel on *tape_device*, making *tape_device* bootable by the firmware.

-r *device* — Sets the root file system device in the boot code (not supported on 68k). Upon booting, the chosen *device* will be mounted by LynxOS as the root file system. *device* may be given as a major-minor pair (*maj*, *min*) or as the name of a block device file. See NOTES regarding device naming with makeboot. The default root device is the device containing the newly bootable *file*. This root device name is used by preboot as the default root device.

EXAMPLES

Making a bootable LynxOS kernel loader program

Make `preboot` bootable:

`makeboot preboot`

Make `preboot` bootable and set the root and boot device to partition A of the SCSI disk at ID 0:

`makeboot -r 6,16 -b /dev/sd2940.0a preboot`

Making bootable CD-ROM media (PowerPC, 68k, and SPARC only)

`makeboot` may be used to make a CD-ROM that contains an ISO9660 file system bootable. `makeboot` must be run on the platform from which the ISO9660 file system will be booted. Before using makeboot, you must create the ISO9660 file system image with `mkisofs -b`. The **-b** option specifies the file name with the ISO9660 image that `makeboot` will make bootable (see `mkisofs(1)`).

The most common use is to make an ISO9660 image that contains `preboot` (the LynxOS kernel loader), and then use `makeboot` to make that `preboot` bootable. This is how Lynx produces the installation product CD-ROM media for PowerPC, 68k, and SPARC systems. For example, assuming that `/tmp` contains 10 MB of files and a copy of `preboot`. To make the ISO9660 image, enter:

`mkisofs -R -b /tmp/preboot isoimage /tmp`

To make this image bootable on PowerPC:

`makeboot -r 2,6 isoimage`

To make this image bootable on 68k:

`makeboot isoimage`

To make this image bootable boot on SPARC:

`makeboot -r 0,6 isoimage`

After making the bootable media, you can use the `cdwrite` utility (currently available only on LynxOS x86 systems) to write the bootable image to a recordable CD-ROM. See `cdwrite(1)` for more information.

Booting a CD-ROM (PowerPC, 68k, and SPARC only)

These examples show how to boot the images created above, assuming that they have been written to a recordable CD-ROM (for example, the LynxOS installation media). These commands are platform-specific. Please refer to your hardware manual for more information.

Power down your system and insert the CD-ROM. At the firmware prompt:

on PowerPC systems, enter:

```
Bug-> pboot 0 60
```

on 68k systems, enter:

```
Bug-> bo 0 60
```

on SPARC systems, enter:

```
ok boot cdrom
```

COMPATIBILITY

On the PowerPC, you do not need to run `makeboot` on the LynxOS kernel after reconfiguring it. This was necessary in the past because the PowerPC platform did not have the `preboot` utility in releases prior to LynxOS 2.5. If you have a system that has both LynxOS 2.4 and LynxOS 2.5 on separate partitions, the `preboot` utility should be used to boot either version of LynxOS. See the `preboot(1)` manual page for more information.

NOTES

LynxOS does not support the El Torrito boot specification for x86 systems. Because of this, LynxOS cannot directly boot CD-ROM media on this platform.

When specifying the minor device of a CD-ROM with makeboot, it must match the physical ID of the CD-ROM device in order to boot directly without user intervention. The default minor number used by LynxOS is 6.

Root and boot device naming is done using heuristic methods which may fail. In particular, if you specify a boot or root device, the device name used by the boot code is arrived at by concatenating the first alphabetic, and the last numeric characters encountered. Thus, `/dev/fd0` will resolve to `f0` and `/dev/fd1200.0` will resolve to `f0`. This may have unexpected results. If you do not specify a device, `makeboot` determines the appropriate device by looking through the `/dev` directory for a device whose major and minor numbers match the one on which file resides. The same naming heuristic is applied to the device name so encountered. Again, errors may occur. If there is any doubt as to which device is required for booting, specify it explicitly.

`makeboot` does not make sure file is in fact an executable program.

CAUTION

Executing `makeboot` will invalidate any existing boot loaders on a disk.

SEE ALSO

Utility Programs - `mkfs, mkisofs, preboot, cdwrite`

NAME

makeinfo – build GNU info files

SYNOPSIS

makeinfo
[-I *dir***] [-D** *var***] [-U** *var***]**
[--error-limit *num* **| -e** *num***]**
[--fill-column *num* **| -f** *num***]**
[--footnote-style *style* **| -s** *style***]**
[--no-headers] [--no-split]
[--no-validate | --no-pointer-validate]
[--no-warn] [--no-number-footnotes | --number-footnotes]
[--output *file* **| -o** *file***]**
[--paragraph-indent *num* **| -p** *num***]**
[--reference-limit *num* **| -r** *num***]**
[--verbose]

makeinfo [--version | -V]

DESCRIPTION

makeinfo is a program for converting texinfo files into *info* files. Texinfo is a documentation language which allows a printed manual and online documentation (an info file) to be produced from a single source file. *info* files contain structured markers to permit easy online navigation of the documentation.

See *Texinfo, the GNU Documentation Format* for details on how to write a texinfo file.

OPTIONS

-I *dir* Adds *dir* to the directory search list for finding files that are included using the '@include' command. By default, makeinfo searches only the current directory.

-D *var* Defines the texinfo flag *var*. This is equivalent to '@set *var*' in the Texinfo file.

-U *var* Makes the texinfo flag *var* undefined. This is equivalent to '@clear *var*' in the Texinfo file.

--error-limit *num* | **-e** *num*
: Sets the maximum number of errors that `makeinfo` prints before exiting (on the assumption that continuing would be useless). The default number of errors that can be printed before `makeinfo` gives up on processing the input file is 100.

--fill-column *num* | **-f** *num*
: Specifies the maximum right-hand edge of a line. Paragraphs that are filled will be filled to this width. The default value for fill-column is 72.

--footnote-style *style* | **-s** *style*
: Sets the footnote style to *style*. *style* should either be '`separate`' to have `makeinfo` create a separate node containing the footnotes for the current node, or '`end`' to have `makeinfo` place the footnotes at the end of the current node.

--no-headers
: Do not generate menus or node headers. This option is useful together with the '**--output** *file*' and '**--no-split**' options to produce a simple formatted file (suitable for printing on a dumb printer) from Texinfo source. If you do not have TeX, these options may allow you to get readable hard copy.

--no-split
: Suppress the splitting stage of `makeinfo`. In general, large output files are split into smaller subfiles, each one approximately 50k bytes. If you specify `--no-split`, `makeinfo` will not split up the output file.

--no-validate | **--no-pointer-validate**
: Suppress the validation phase of `makeinfo`. Normally, after the file is processed, some consistency checks are made to ensure that cross references can be resolved, etc.

--no-warn
: Suppress the output of warning messages. This does *not* suppress the output of error messages, simply warnings. You might want this if the file you are creating has texinfo examples in it, and the nodes that are referenced don't actually exist.

--no-number-footnotes
Supress the automatic numbering of footnotes. The default is to number each footnote sequentially in a single node, resetting the current footnote number to 1 at the start of each node.

--number-footnotes
Explicitly request that footnotes be numbered, as they are by default.

--output *file* **|** **-o** *file*
Write the output in the *file* specified, instead of in the name specified in the texinfo file itself. *file* can be the special token '–', which specifies standard output.

--paragraph-indent *num* **|** **-p** *num*
Sets the paragraph indentation to *num*. The value of *num* is interpreted as follows:

A value of 0 means don't change the existing indentation at the starts of paragraphs.

A value less than zero means indent paragraph starts to column zero by deleting the existing indentation.

A value greater than zero is the number of spaces to leave at the front of each paragraph start.

--reference-limit *num* **|** **-r** *num*
Issue a warning if any single node is referenced more than *num* times in the info file. The default *num* is 1000.

--verbose Causes `makeinfo` to inform you as to what it is doing. Normally `makeinfo` only outputs text if there are errors or warnings.

--version | -V Prints out the version number of `makeinfo`.

SEE ALSO

Utility programs - `info`

Documents - the '`texinfo`', '`info`', and '`makeinfo`' info nodes;

Texinfo, the GNU Documentation Format, by Robert Chassell and Richard Stallman.

AUTHOR

Brian Fox wrote and maintains `makeinfo`.

COPYING

NAME

man – format and display the on-line manual pages

SYNOPSIS

man [–adfhkKtwW] [–m *system***] [–p** *string***] [–C** *config_file***] [–M** *path***]**
[–P *pager***] [–S** *section_list***] [***section***]** *name* ...

DESCRIPTION

man formats and displays the on-line manual pages. This version knows about the MANPATH and (MAN)PAGER environment variables, so you can have your own set(s) of personal manpages and choose whatever program you like to display the formatted pages. If *section* is specified, man only looks in that section of the manual. You may also specify the order to search the sections for entries and which preprocessors to run on the source files via command line options or environment variables. If *name* contains a / then it is first tried as a file name, so that you can do man ./foo.5 or even man /cd/foo/bar.1.gz.

To see a list of all available manpages in the standard LynxOS distribution, use the more or less commands to view the text file /usr/man/whatis.

OPTIONS

–C *config_file* Specify the man.config file to use; the default is /usr/lib/man.config. (See man.config(5).)

–M *path* Specify the list of directories to search for manpages. If no such option is given, the environment variable MANPATH is used. If no such environment variable is found, the default list is found by consulting /usr/lib/man.config. An empty substring of MANPATH denotes the default list.

–P *pager* Specify which pager to use. This option overrides the MANPAGER environment variable, which in turn overrides the PAGER variable. By default, man uses /bin/more.

-S *section_list*	List is a colon separated list of manual sections to search. This option overrides the `MANSECT` environment variable.
-a	By default, `man` will exit after displaying the first manual page it finds. Using this option forces `man` to display all the manual pages that match *name*, not just the first.
-c	Reformat the source manpage, even when an up-to-date cat page exists. This can be meaningful if the cat page was formatted for a screen with a different number of columns.
-d	Don't actually display the manpages, but do print gobs of debugging information.
-D	Both display and print debugging info.
-f	Equivalent to `whatis`.
-h	Print a one line help message and exit.
-k	Equivalent to `apropos`.
-K	Search for the specified string in *all* manpages. Warning: this is probably very slow! It helps to specify a section.
-m *system*	Specify an alternate set of manpages to search based on the system name given.
-p *string*	Specify the sequence of preprocessors to run before `nroff` or `troff`. LynxOS currently does not contain the `nroff` or `troff` preprocessor needed to use this option.
-t	Use `/usr/bin/groff -Tps -mandoc` to format the manual page. LynxOS currently does not contain the `groff` preprocessor needed to use this option.
-w or **-path**	Don't actually display the manpages, but do print the location(s) of the files that would be formatted or displayed. If no argument is given: display (on `stdout`) the list of directories that is searched by `man` for manpages.

-W	Like **-w**, but print file names one per line, without additional information. This is useful in shell commands like `man -aW man	xargs ls -l`.

CAT PAGES

`man` will try to save the formatted manpages, in order to save formatting time next time these pages are needed. Traditionally, formatted versions of pages in `DIR/manX` are saved in `DIR/catX`, but other mappings from `man` directory to `cat` directory can be specified in `/usr/lib/man.config`. No catpages are saved when the required `cat` directory does not exist.

It is possible to make `man` suid to a user man. Then, if a cat directory has owner man and mode 0755 (only writable by man), and the cat files have owner man and mode 0644 or 0444 (only writable by man, or not writable at all), no ordinary user can change the cat pages or put other files in the cat directory. If `man` is not made suid, then a cat directory should have mode 0777 if all users should be able to leave cat pages there.

The option **-c** forces reformatting a page, even if a recent cat page exists.

ENVIRONMENT

`MANPATH`	If `MANPATH` is set, its value is used as the path to search for manual pages.
`MANROFFSEQ`	If `MANROFFSEQ` is set, its value is used to determine the set of preprocessors run before running nroff or troff. Currently not supported in LynxOS.
`MANSECT`	If `MANSECT` is set, its value is used to determine which manual sections to search.
`MANWIDTH`	If `MANWIDTH` is set, its value is used as the width manpages should be displayed. Otherwise the pages may be displayed over the whole width of your screen.
`MANPAGER`	If `MANPAGER` is set, its value is used as the name of the program to use to display the manpage. If not, then `PAGER` is used. If that has no value either, `/bin/more` is used.

LANG	If LANG is set, its value defines the name of the subdirectory where man first looks for manpages. Thus, the command LANG=dk man 1 foo will cause man to look for the foo manpage in .../dk/man1/foo.1, and if it cannot find such a file, then in .../man1/foo.1, where ... is a directory on the search path.
NLSPATH, LC_MESSAGES, LANG	The environment variables NLSPATH and LC_MESSAGES (or LANG when the latter does not exist) play a role in locating the message catalog. (But the English messages are compiled in, and for English no catalog is required.) Note that programs like col(1) called by man also use e.g. LC_CTYPE.
PATH	PATH is used in the construction of the default search path for manpages.
SYSTEM	SYSTEM is used to get the default alternate system name (for use with the **-m** option).

SEE ALSO

Utility Programs - apropos, whatis, less

NAME

`mem` – memory dumping utility

SYNOPSIS

mem [*format***] [-f** *memdev***] [-n]** *addr size*

DESCRIPTION

`mem` dumps *size* bytes of memory starting at the physical address *addr* in a format specified by *format*. If no format is specified, the memory is dumped in raw form.

The *addr* argument specifies the physical address where dumping is to begin. It can be specified in decimal, octal, or hexadecimal, depending on the prefix: No prefix means decimal; a leading “`0`” means octal; and a leading “`0x`” means hexadecimal. The *size* argument specifies the number of bytes of memory to dump. It can be specified in decimal, octal, or hexadecimal in the same manner as *addr*.

OPTIONS

The *format* argument can be one of the following:

-a Display bytes as ASCII characters. The parity (high) bit is ignored. Characters which are not displayable, including the space character, are displayed as ascii names (null = `nul`, newline = `nl`, carriage return = `cr`, space = `sp`, etc.).

-c Display bytes as printable characters, as they would appear in C strings. Characters which are not displayable are displayed as C escapes (null = `\0`, newline = `\n`, etc.). Other undisplayable characters appear as 3-digit octal integers. Any bytes with the parity bit set are displayed in octal.

-d Display bytes in decimal.

-o Display bytes in octal.

-x Display bytes in hexadecimal.

If no format is specified, the memory contents are dumped unmodified (i.e., raw) to the standard output channel.

If a format is specified, the memory is dumped 16 bytes per line with the beginning address of each line at the left. By default, `mem` aligns the formatted output so that each line begins on a 16-byte boundary, regardless of whether or not *addr* is on a 16-byte boundary. If *addr* is not on a 16-byte boundary, then the first line of output will show dashes for those memory locations that are less than *addr* to indicate that they have not been read. The **-n** option overrides the 16-byte boundary alignment, so that the address of the first line is *addr*, the second line is *addr+16*, and so on.

If more than one format is specified, only the last one is used; the others are ignored.

By default, `mem` opens and reads from the memory device `/dev/mem`. The **-f** option can be used to specify *memdev* as an alternate memory device file. In either case, `mem` prints a warning message if the memory device file is not a character special file.

SEE ALSO

Utility Programs - `flook, od`

NAME

`merge` – three-way file merge

SYNOPSIS

`merge [ -p ]` *file1 file2 file3*

DESCRIPTION

`merge` incorporates all changes that lead from *file2* to *file3* into *file1*. The result goes to standard output, if **`-p`** is present, otherwise into *file1*. `merge` is useful for combining separate changes into an original. Suppose *file2* is the original file and both *file1* and *file3* are modified versions of *file2*, then `merge` will combine both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. `merge` prints how many overlaps occurred and includes both alternatives in the result. The alternatives are delimited as follows:

<<<<<<< *file1*

`<lines in` *file1*`>`

=======

`<lines in` *file3* >

>>>>>>> *file3*

If there are overlaps between *file1* and *file3*, the user should edit the result and delete one of the alternatives.

SEE ALSO

Utility Programs - `diff3, diff, co`

NAME

`mibIIagent` – SNMP MIB-II subagent daemon

SYNOPSIS

`/net/mibIIagent`

DESCRIPTION

This Simple Network Management Protocol (SNMP) subagent provides support for the following MIB-II sub-groups:

```
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).interfaces(2)
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).at(3)
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).ip(4)
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).icmp(5)
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).tcp(6)
iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).udp(7)
```

This subagent is invoked from the `/etc/config/netservice.snmp` script. The SNMP installation script configures the `/etc/sysconfig` file to automatically begin this subagent.

SEE ALSO

Utility Programs - `snmpd`, `lynxos_agent`

File Formats - `config.snmp`

NAME

`mine` – minesweeper game

SYNOPSIS

/usr/games/mine [-kq] [*level***]**

/usr/games/mine [-kq] [*width height bombs***]**

DESCRIPTION

The `mine` program is a version of the popular minesweeper game. It runs on the x86 version of LynxOS using VGA graphics. It has 9 difficulty levels, numbered 1 through 9, plus a custom level that can be created by the user. This game is best played with a mouse, although a keyboard interface is provided using the **-k** option.

There are two different ways to win a game:

1. Clear all non-bomb squares.
2. Flag all the bombs.

The method of flagging all the bombs can be very handy when a guessing situation arises near the end of a game.

The `mine` program maintains a local high score list as well as an all-time high score list. The all-time high score list is built using all of the high scores files in the directory specified by the `MINEREMOTE` environment variable.

OPTIONS

-k Use the keyboard instead of the mouse.

-q Disable question marks.

ENVIRONMENT VARIABLES

The `mine` program makes use of the following environment variables:

`HOME` Directory where the local high score file is saved.

`MINEREMOTE`	Directory where remote high score files are saved. This directory would typically reside on an NFS-mounted file system.

FILES

`/dev/mouse`	Mouse node
`$HOME/bestminers`	Local high score file
`$MINEREMOTE/hi.`*host*`.`*user*	Remote high score file, where *host* is the name of the local machine, and *user* is the name of the user.

NAME

`mkcontig` – make a contiguous file

SYNOPSIS

`mkcontig` *name size*

DESCRIPTION

`mkcontig` uses the `mkcontig` system call to create the contiguous file *name* with the given *size*. The system locates (if possible) a contiguous extent of the requested size.

The *size* value can be suffixed with **`m`**, meaning megabytes, **`k`**, meaning kilobytes, or **`b`**, meaning 512-byte blocks. The `mkcontig` system call always rounds the size up to a multiple of the file system's block size.

EXAMPLE

To make a 20MB swap file called `/swap`, type the following line at the command line prompt:

```
bash$ mkcontig /swap 20m
```

SEE ALSO

System Calls - `mkcontig()`

NAME

mkdir – create directories

SYNOPSIS

mkdir [-pf] *directory* ...

DESCRIPTION

mkdir creates directory files with the given pathnames. The standard entries "." and ".." are automatically created in the new directory. mkdir must have write permission into the parent directory of the new directories.

OPTIONS

-f Suppresses error messages.

-p Causes mkdir to create each non-existent directory along each given pathname. Thus the command:

```
mkdir -p test1/data/numlists/integers
```

will create the subdirectory integers along with test1, data, and numlists, if any of those do not exist.

SEE ALSO

Utility Programs - rmdir

System Call - mkdir()

NAME

`mkfs` – make a file system

SYNOPSIS

mkfs [-ovS] [-b *blocksize* **]** *device* **[[/]***inodes***]**

DESCRIPTION

`mkfs` creates the data structures which comprise a LynxOS file system on the indicated *device*. *device* should be a raw device. All data previously on the device are lost.

The amount of space on the *device* is determined automatically. The size of the inode table (in terms of *inodes*, not bytes) may be given as *inodes*. If *inodes* is preceded by a slash, then it represents an estimate of the average file size (in blocks) on the device; the size of the inode table is set accordingly. `mkfs` will create one inode for every sixteen free blocks if `inodes` is not given.

OPTIONS

-o Instructs `mkfs` to create a file system with the old pre-2.3.0 style free block list. Specifying **-o** without **-b** blocksize or with blocksize set to 512, will create a file system that is compatible with versions of LynxOS between 1.2.1 and 2.2.1. It is no longer possible to make file systems that are compatible with pre-1.2.1 systems.

-v Causes `mkfs` to report the size of the device and the inode table. The file system is created as usual.

-S Instructs `mkfs` to create a super block with appropriate values for the device and inode table sizes, then exit without affecting other parts of the file system. This operation may be useful in desperate situations, such as when `fsck` reports that the super block is “hopelessly damaged”.

-b *blocksize* Causes `mkfs` to create a file system with *blocksize* byte blocks. *blocksize* can be 512, 1024, 2048, 4096, 8192, 16384, or 32768.

SEE ALSO

Utility Programs - `fsck`, `makeboot`, `mkpart`

File Formats - `boot`, `superblock`

NAME

`mkimage` – make an image of a LynxOS kernel

SYNOPSIS

mkimage [-v?] *spec_file outfile*

DESCRIPTION

`mkimage` creates an image file of a LynxOS kernel and file system. This image file can be stored in ROM for ROM booting or kept on a remote server for network booting via `tftp`.

The image file created by `mkimage` contains a LynxOS kernel, file system, and (optionally) resident text segments. The file system, containing a basic set of initial files, is appended to the kernel and is accessed by the LynxOS ramdisk driver as the root file system. The optional text segments of all executables are written at the end of the image file at a page boundary.

The image file is written in normal `a.out` format and contains enough information for the LynxOS kernel to configure itself at run time.

OPTIONS

-? Displays help message of all available options.

-v Verbose. show progress of this utility.

spec_file Specification file containing directives for the generation of the ROMable kernel. See the `mkimage.spec(5)` manual page for details about this file.

outfile Resultant kernel image file.

At the end, `mkimage` will summarize the static and runtime memory map of ROMable kernel image along with the sizes of various segments. The static map provides the addresses of various segments as they appear in ROM/disk image. The runtime memory map provides addresses of various segments as they would appear in kernel address space at run time.

SEE ALSO

Utility Programs - `mkramdisk`

File Formats - `mkimage.spec`

BUGS

SPARC: Due to defects in the microSPARC chip, `mkimage` does not allow any portion of the kernel (kernel, file system or resident text) to be run in ROM. That is all of **`ostext=`**, **`text=`**, and **`root=`** must be RAM and are changed to that if specified otherwise.

SPARC: The resident text does not work in this release.

NAME

`mkisofs` – Create an ISO9660 or ISO9660 + RRIP file system

SYNOPSIS

mkisofs [-R] [-b bootfile] *dest file1*, *file2*, ... *filen*

DESCRIPTION

`mkisofs` creates an ISO9660 or ISO9660 + Rock Ridge Interchange Protocol (RRIP) file system from the input files specified. ISO9660 will be used throughout this man page to mean either an ISO9660 or an ISO9660 + RRIP file system.

The file system is created in *dest*. If *dest* is a file or a disk partition, that file or partition will contain an ISO9660 file system. If *dest* is copied directly to a CD or disk partition, then the target can be mounted.

NOTE: LynxOS supports mounting a partition that contains an ISO9660 file system.

The first 16 logical 2 KB blocks (0 - 15) are reserved by LynxOS. This area will be used by `makeboot` to make a bootable ISO9660 file system.

Logical 2 KB block 16 on an ISO9660 file system contains the Primary Volume Descriptor (PVD). Within the PVD there are 512 bytes of reserved space for application use. LynxOS reserves the first 256 bytes of this space for operating system use. The second 256 bytes is available for customer applications. Refer to `<iso9660.h>` for details.

`mkisofs` is usually used in conjunction with `cdwrite` utility (currently supported on x86 systems only). `mkisofs` creates the ISO9660 file system that `cdwrite` writes to a CD Recorder (CD-R). It is an expensive development cycle to use `mkisofs` followed by `cdwrite` of CD-R media, then a mount of the media to check things out. If dest for `mkisofs` is a partition of a disk, the partition can be mounted and examined for correctness without using CD-R media. Once the image is satisfactory, you can use `cdwrite` to write the image to a CD-R.

OPTIONS

-R The Rock Ridge Interchange Protocol (RRIP) flag. When thrown, an ISO9660 with RRIP extensions file system will be created. Without this option, just an ISO9660 file system will be created.

-b bootfile The **bootfile** is an absolute path into the volume created. If found, information about **bootfile** will be stored in the LynxOS Reserved Area of the Primary Volume Descriptor for `makeboot`. This provides `makeboot` with the information needed to make a bootable ISO9660 volume. See `makeboot(1)` for more information.

EXAMPLES

```
mkisofs -R /tmp/targetfile *
mkisofs -R /tmp/targetfile src1 src2 src3
mkisofs -R /dev/rhd0b *
```

BUGS

`mkisofs` uses a lot of memory. For 23000 files, `mkisofs` uses 10 to 12 MB of memory. You may have to make adjustments in `/etc/starttab` or start up virtual memory to run this utility.

The **-R** flag creates a RRIP directory entry. This entry is limited to 256 bytes. If the entry exceeds 256 bytes, `mkisofs` will quit with an error. Directory entries can exceed 256 bytes if very long file names or very long symbolic links are present.

If there is a permission problem when reading NFS files, `mkisofs` will exit with an error. It should just pad the file with zeros and print out a diagnostic message.

NAME

`mknod` – create a special file entry

SYNOPSIS

mknod *name type major minor* **[** *perm* **]** ***where type is c or b***

mknod *name type* **[** *perm* **]** ***where type is f or p***

mknod –a [*nodetab* **]**

DESCRIPTION

`mknod` creates a directory entry and an inode for a special file. The entry can be placed in any directory, although by convention commonly used special files are placed in the `/dev` directory. The type of the node should be **c** for a character device, **b** for a block device, or **f** or **p** for a named pipe (FIFO). The *major* number of the node should correspond to the device number of the installed device. The *minor* number should be the appropriate "subdevice" number, such as the number of a particular SCSI device.

perm is the permissions, in octal, which the node should be created with. If perm is not given the node will be created with permissions 0666. *perm* is modified by the umask environment variable. It may be necessary to set the umask variable to zero before using `mknod`, in order to set the permissions correctly.

The **–a** option instructs `mknod` to read a list of special file descriptions from a given *nodetab* file. If *nodetab* is not specified, the file `/etc/nodetab` is read by default. Usually, node tables are generated automatically by the `config` utility during the course of building a LynxOS kernel. Each line of the node table has the form

```
name:type:major:minor:perm
```

The *type* will be either `c`, `b`, or (rarely) `f`. Additionally, the type field can be used to create symbolic links to other nodes: if *type* is a string beginning with the "`@`" character, a symbolic link from *name* to that string (minus the "`@`") will be created when the table is processed.

By convention, filenames for nodes in a node table are given relative to the current directory. Therefore, invocation of `mknod` with **-a** should be done in the directory where the nodes are to be placed (often `/dev`).

NOTE

Only one node should be created for each device.

Permissions for nodes are modified by the `umask` of the process. This may result in a node permission different from the expected mode. This is in accordance with standard UNIX behavior.

SEE ALSO

Utility Programs -	`devices, devinstall, sh, dlsh, bash, csh`
System Calls -	`mknod(), umask()`
File Formats -	`nodetab`

NAME

`mkpart` – make and/or administer partitions on a disk

SYNOPSIS

`mkpart` *device_name*

DESCRIPTION

NOTE: If you use LynxOS for SPARC, please see the `mkpart_sparc(1)` manual page.

`mkpart` is used to make partitions on the disk and update bad block information on the disk. The argument to `mkpart` is the device name (without any partition as in `setactive`) *device_name*.

`mkpart` supports two types of disks—those where partitions are based on cylinders and heads (IDE disks) and those where partitions are based on blocks (SCSI disks).

When a partition is created with `mkpart`, the partition is given the Lynx partition ID (64). This is not compatible with DOS which uses (6) as the partition ID. In order to create a partition with a partition ID different from the default Lynx ID, the **`m`** command with the `/` option must be used.

For example, in order to create three partitions—one for DOS, one with ID17, and one for LynxOS—the following command would be used:

`m` *size1*`/` *size2*`/17` *size3*

where *size1*, *size2*, and *size3* are the appropriate sizes for each partition. On SCSI disks, sizes should be given in megabytes; on non-SCSI disks, sizes should be given in cylinders. On non-SCSI disks, an *x* megabyte partition can be created by dividing *x* by the number of megabytes per cylinder (given by `mkpart`) and using that value as the partition size. Note that when using the `m` command with the `/` option, if the `/` is not immediately followed by a number, then the partition ID defaults to 6 (DOS partition).

In order to create a third partition on a disk which already has two partitions, the **m** command can be used as follows:

```
m - - all
```

This will create a third, LynxOS partition which will use all the blocks (or cylinders) that have not already been included in a partition, without changing the first two partitions. The / option can be combined with a – and `all` to change the partition ID.

The commands available for disks partitioned by blocks (SCSI disks) are:

p Print partition information.

m *size size size size*
Make partitions.

q Quit and don't change current partition information.

? or **h** Print the help menu.

e or **ctrl-D** Exit and save the partition information entered.

The commands available for disks partitioned by cylinders and heads (IDE disks) are:

c Clear the bad block list.

f Find all blocks marked bad.

A <1-4> Set the active (boot) partition.

a blk Add to the bad block list.

t cyl head Add blocks on track to bad block list.

b Print bad block list.

p Print partition information.

m *cyls cyls cyls cyls*
Make partitions.

s *pathname* Save bad block list in the file *pathname* specified.

q Quit and don't change bad block list.

? or **h**	Print the help menu.
e or **ctrl-D**	Exit and save the partition information entered.

DIAGNOSTICS

`mkpart` fails for the following reasons:

- If the specified *device_name* does not exist.
- If the specified *device_name* is not a disk.
- Wrong usage or failure to specify required parameters.
- If the specified block/track to be added does not exist.
- If the specified file *pathname* cannot be opened for saving the bad block list.
- If all the disk blocks have been used by initial partitions and the command asks that another partition be added.

WARNING

Partition A on the PowerPC is reserved by LynxOS for booting purposes (in the LynxOS 2.5 release). It should be created with a size of zero in `mkpart`. For example, to make three partitions on a 1 GB SCSI disk:

```
mkpart /dev/sd0
Command? m 0 300 300 all
```

The `makeboot` utility uses the zero length partition A when creating bootable files. If you have an existing partition A that contains user data, it will be inaccessible when the `makeboot` command is executed.

SEE ALSO

Utility Program - `setactive`

NAME

`mkpart` – label and partition a disk (SPARC)

SYNOPSIS

mkpart [-S [-n *ncyl* **-t** *ntrk* **-s** *nsec***]]** *device*

DESCRIPTION

`mkpart` is used to initialize a hard or floppy disk for use under LynxOS. Floppy disks must be formatted (via `fmtflop` under LynxOS, or some other means of your choice) before `mkpart` is used. SCSI disks generally require no low-level formatting before `mkpart` is used.

`mkpart` installs a disk label in block `0` (cylinder `0`, track `0`, sector `0`) of the given device. This disk label contains various information about the disk that is essential to its use under LynxOS. In particular, a partitioned disk (one with multiple file systems on it) must have a disk label.

`mkpart` operates in a menu-driven fashion. Here are the available commands:

`q` Quits `mkpart` without modification to your disk. Use this option if you make some mistake you wish to undo.

`p` Prints the current partitioning of the disk.

`m` Creates a new partition table. LynxOS supports up to four partitions per disk. The `m` command is given four additional numbers to indicate the size of the four desired partitions. These are described under “Making Partitions” below.

`e` Writes the disk label onto the device and exits `mkpart`. Use this command when the partition table is as you wish.

LynxOS-Style Disk Labeling

Disks that are not to be booted on a SPARC system can have the standard Lynx-style disk labels. These labels are produced by `mkpart` when it is invoked with no parameters other than the device name. Partitions on a Lynx-style labeled disk are specified in numbers of blocks. Each block is 1/2 of a kilobyte.

SunOS-Compatible Disk Labeling

A disk to be booted on a SPARC system running LynxOS *must* be partitioned. It is not possible to boot from an unpartitioned disk from the Sun Open Boot PROM.

Furthermore, the disk must be labeled with a disk label that is comprehensible to Sun's Open Boot PROM. `mkpart` allows you to create a SunOS-compatible label by specifying the **-S** option on the command line.

If **-S** is specified, then `mkpart` also must know the number of cylinders, heads, and sectors on the disk. If there is already a SunOS-compatible disk label on the disk, then `mkpart` will be able to retrieve the numbers from that label. Otherwise, you must specify the appropriate numbers using the **-c**, **-t**, and **-s** options. `fmtscsi` **-v** can be used to determine the number of cylinders, heads, and sectors on a SCSI disk.

When a SunOS-compatible label is being created, `mkpart` creates partitions consisting of an integral number of cylinders. In this case, the `m` command takes numbers of cylinders to determine the size of the partitions to create. To determine the number of megabytes in a cylinder, multiply the number of tracks times the number of sectors and divide by 2048.

Making Partitions

The `m` command takes up to four partition sizes. If you want less than four partitions on your disk, simply specify the appropriate sizes for all by the last partition you want, and then specify `all` for your final partition. This causes `mkpart` to assign all remaining space on the disk to the final partition.

When creating a LynxOS-style label, partition sizes are given in numbers of 512-byte blocks. When creating SunOS-compatible labels, partition sizes are given as numbers of cylinders.

An easier method for creating appropriately-sized partitions with a SunOS-compatible label is to make partitions with some arbitrary number of cylinders, see how many megabytes are in each partition (using the `p` command), and then modify your cylinder numbers accordingly.

It is generally a good idea to type `all` for the last partition.

EXAMPLES

To invoke `mkpart` on SCSI disk 0 so as to create a SunOS-compatible label, type:

```
mkpart -S /dev/rsd0
```

(Note: a disk will have a SunOS-compatible label if you bought it from Sun Microsystems, or if you have already written an appropriate label on it with `mkpart` **-S**.)

To invoke `mkpart` on disk 3, which has 1310 cylinders, 42 sectors and 7 heads, type:

```
mkpart -S -n 1310 -t 7 -s 42 /dev/rsd3
```

To place a single large partition on a disk (either LynxOS- or SunOS-labeled), type

```
Command? m all
```

To place one partition of 500 blocks or cylinders on the disk, followed by a second partition containing the remainder of the disk, type

```
Command? m 500 all
```

To place four partitions of various sizes on the disk, type

```
Command? m 313 272 500 all
```

To partition a floppy disk so that it will be bootable on a SPARCsystem, invoke `mkpart` with

```
mkpart -S -c 80 -t 2 -h 18 /dev/rfd0
```

Then, create a single partition with

```
Command? m all
```

and write the disk label with

```
Command? e
```

ERRORS

If `mkpart` is invoked with the **-S** option, and is unable to determine the appropriate number of cylinders, sectors, and heads, it will complain and exit.

SEE ALSO

Utility Programs - `fmtscsi`, `mkfs`, `makeboot` (SPARC version), `openboot`, `sparcboot`, `fmtflop`

NAME

mkramdisk – dynamically install a ramdisk device

SYNOPSIS

mkramdisk [-b *bdrv***] [-c** *cdrv***] [-n** *bname***] [-r** *cname***]** *blocks*

DESCRIPTION

mkramdisk installs the requisite “devices” and makes nodes for block and raw interfaces to a memory resident “disk”. The device can be used exactly as one would use a floppy or hard disk. A file system can be placed on the disk (with mkfs) and it can then be mounted, just like a real disk.

The size of the ramdisk is given in 512-byte blocks and there should be enough available physical memory.

This program assumes that the ramdisk driver has been configured into the operating system.

OPTIONS

-b *bdrv* — The name of the block interface to the ramdisk driver, as presented to the kernel upon driver configuration. mkramdisk assumes in the absence of this option that the name is RD.

-c *cdrv* — The name of the character interface to the ramdisk driver, as presented to the kernel upon driver configuration. The default name is RRD.

-n *bname* — The node name mkramdisk should use for the block interface to the ramdisk. The default name is /dev/rd0.

-r *cname* — The node name mkramdisk should use for the character (raw) interface to the ramdisk. The default name is /dev/rrd0.

NOTE

The ramdisk driver will only support one installed ramdisk at a time; once `mkramdisk` has been run, another ramdisk cannot be created.

SEE ALSO

Utility Programs - `mkfs`

NAME

`mkshlib` – create a shared library (x86, 68k, SPARC)

SYNOPSIS

mkshlib -s *specfil* **-t** *target* **[-h** *host***] [-n] [-L** *dir ...***] [-q]**

DESCRIPTION

`mkshlib` builds shared libraries in the manner of System V UNIX. A shared library functions like an ordinary non-shared archive library with the following important difference: programs linked with a shared library will share the library code during execution, while programs linked with a non-shared library will get their own copies of the library routines they use.

A shared library has two components — the host-shared library and the target-shared library. The host-shared library is an ordinary archive file (see `ar`) that is used to link-edit programs with the shared library. It is a true archive file, which means that it can be used like any other non-shared archive library.

The target-shared library has the form of an executable COFF object file. Whenever a program that uses the shared library is executed, the operating system maps the target-shared library's text into the address space of that process and creates a separate copy of the target-shared library's data for that process. The text is shared, but the data is not. The code for a target-shared library is not kept in core when no process is using it.

`mkshlib` builds the shared library according to its command line options and the contents of the shared library specification file, *specfil*. The command line options are described in the next section. The shared library specification file outlines the construction of both the host- and target-shared libraries. It is organized using a simple grammar that uses directives. Some directives are one-line directives and some are multi-line. Multi-line directives are followed by one or more specification lines and remain "in effect" until the next directive or until the end of the file.

#address *sectname address*

Specifies that section, *sectname*, in the target shared library will start at address, *address*. This directive must always be specified for the `.text` section, and it must be specified for the `.data` section if the `.data` section is not empty, or if the `.bss` section is not empty and the `.bss` section is not given its own starting address directive. If the `.bss` section is not given its own starting address directive, then the `.bss` section will start at the end of the `.data` section. See “Shared Libraries” in the LynxOS User’s Guide for a list of standard addresses.

#target *pathname*

Specifies that the target-shared library is expected to reside in file *pathname* on the *target* machine. This is the pathname that the operating system uses to locate the target-shared library file when executing programs that use this shared library. This pathname will usually be equal to the *target* given with the **-t** command line option, but it doesn't have to. The difference is that the **#target** directive specifies the pathname that actually gets recorded into the host-shared library (and consequently into any program that links with it), while the **-t** option specifies the name of the target-shared library file created by `mkshlib`. This allows the target-shared library file created by `mkshlib` to be moved to a different pathname, perhaps on a different system. In order to be used, of course, the target-shared library file must end up with a pathname that exactly matches the **#target** directive’s *pathname*.

This directive must be specified exactly once.

#branch

Specifies the beginning of a multi-line branch table specification. Each line of the branch table specification has the following format:

funcname `<whitespace>` *position*

where *funcname* is the name of an external function symbol and *position* is a positive integer or range of positive integers specifying a slot or range of slots in the branch table. A range has the form *position1-position2*. The only restriction on the branch table is that it must have *exactly one* symbol associated with each position between one and the highest specified position, inclusive.

In addition to using a range of positions, a symbol may be associated with more than one position by specifying it on more than one specification line. Whatever the case, if a symbol is associated with more than one position, then it is defined to have the address of the slot in the branch table corresponding to the highest such position. All other slots associated with that symbol are “dummy” slots that are not used. In future versions of the shared library, these “dummy” slots can be used for new function symbols. This directive must be specified exactly once.

#objects Specifies the beginning of a list of object files used to create the host- and target-shared libraries. The object files in this list are separated by white space, so that several may be specified per line. The ordering of this list is significant, because it is used in both linking the target-shared library and creating the host-shared library. The **#objects** directive must be separated from the list of object files by a newline. This directive must be specified exactly once.

#objects noload

Specifies the beginning of a list of host-shared libraries to be searched when resolving symbols not defined in this target-shared library. These libraries are searched in the order given. The host-shared libraries in this list are separated by white space, so that several may be specified per line. Each host-shared library can be specified either by its full pathname or in the form **-lX**, where `/lib/libX.a` is the name of the host-shared library. This shorthand notation works the same way as for `gcc`. This directive may be specified any number of times.

#hide linker [*]

Specifies that certain symbols in the host-shared library's object files that would normally be `external` shall be made `static` instead. If the optional “*****” is specified, then the directive is a one-line directive, and it causes all `external` symbols to be made into `static` symbols. Otherwise, the directive is followed by one or more lines, each specifying one or more patterns separated by white space. Each `external` symbol that matches any of the patterns is made static.

The `#hide` directive has no effect on symbols specified in the `#init` or `#branch` directives. Such symbols are always `external`.

This directive may be specified any number of times.

#export linker [*]
Specifies that certain external symbols in the host-shared library's object files that would otherwise have been switched to static by a `#hide` directive, should nonetheless remain external. This directive is useful when it is desirable to "hide" all but a small list of symbols. For example:

```
#hide linker *
#export linker
symbol1
symbol2
```

causes all external symbols to be made `static` except `symbol1`, `symbol2`, and of course those specified in the `#init` or `#branch` directives.

This directive may be specified any number of times.

#init *object*
Specifies the beginning of a multi-line initialization specification for the object file, *object*. Each line of an initialization specification has the following format:

ptr `<whitespace>` *import*

where *ptr* is a pointer to the imported symbol, *import*. The symbol, *ptr* must be defined in the object file, *object*. `mkshlib` generates assembly code for each such line that is equivalent to the following C code:

```
ptr = &import;
```

This directive may be specified at most once per object file.

#ident *string*	Specifies that a `.comment` section, containing the string, *string*, is to be added to the target-shared library and to each object file in the host-shared library.
##	Specifies a comment that extends to the end of the line. `mkshlib` ignores all comment lines.

OPTIONS

-h *host*	Specifies that *host* is the name of the host-shared library file to be created by `mkshlib`. The host-shared library will not be created if this option is not given.
-L *dir ...*	Specifies the directories to be searched ahead of the default directories when searching for other host-shared libraries given in an `#objects noload` directive. If this option is specified more than once on the command line, then the resulting list of directories will be searched in the order given before searching any of the default directories.
-n	Specifies that the target-shared library shall not be created. In this, case, the **-t** option is used to specify the pre-existing target-shared library file from which to build the host-shared library.
-s *specfil*	Specifies the shared library specification file, *specfil.* See description above.
-t *target*	Specifies that *target* is the name of the target shared library file to be created by `mkshlib`, or, if the **-n** option is given, that *target* is the name of the target-shared library file from which to build the host-shared library.
-q	Suppress (quiet) any warning messages.

SEE ALSO

Utility Programs -	`gcc, ld, ar`
File Formats -	`coff`

NAME

mktimestamp – embed time stamp in a string

SYNOPSIS

mktimestamp [*string* **]**

DESCRIPTION

mktimestamp gets the current system time, embeds it in *string* if provided, and writes the result to standard output. If string is absent the time is written out by itself. If *string* is provided, then each percent sign (%) in the *string* is replaced by the time.

mktimestamp is most often used in makefiles to produce a timestamp function. The program being made can include a call to the function in its startup routines to indicate to the user the date it was created. A typical use in a makefile might be as follows:

```
mktimestamp "TS(){printf("created %n");}" > timestamp.c
$(CC) -c timestamp.c
rm -f timestamp.c
$(CC) -o target timestamp.o $(OBJECTS) $(LFLAGS)
```

The above program would include a call to TS() the beginning.

SEE ALSO

Utility Programs - make, cc

NAME

`more` – file perusal filter for crt viewing

SYNOPSIS

more [-cdflsu] [-*n*] [+*linenumber*] [+/*pattern*] [*name ...*]

DESCRIPTION

`more` is a filter which allows examination of continuous text, one screenful at a time on a soft-copy terminal. It normally 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, another screenful is displayed. Other possibilities are enumerated later.

OPTIONS

The command line options are:

-n An integer which is the size (in lines) of the window which `more` will use instead of the default.

-c `more` will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while `more` is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

-d `more` will prompt the user with the message `Press space to continue, 'q' to quit` at the end of each screenful, and will respond to subsequent illegal user input by printing `Press 'h' for instructions` instead of ringing the bell. This is useful if `more` is being used as a filter in a setting, such as a class, where many users may be unfamiliar with `more`.

-f This causes `more` to count logical lines, rather than screen lines. That is, lines that are not folded.

-l Do not treat control-l (form feed) in a special way. If this option is not given, `more` will pause after any line that contains a control-l, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

-s Squeeze multiple blank lines from the output, producing only one blank line.

-u Normally, `more` will handle underlining in a manner appropriate to the particular terminal: if the terminal can perform underlining or has stand-out mode, `more` will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **-u** option suppresses this processing.

+*linenumber* Start up at *linenumber*.

+/*pattern* Start up two lines before the line containing the regular expression *pattern*.

`more` looks in the file `/etc/termcap` to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

`more` looks in the environment variable `MORE` to pre-set any flags desired. For example, if you prefer to view files using the `-c` mode, the `csh` command:

```
setenv MORE -c
```

or the `sh` command sequence:

```
MORE='-c'; export MORE
```

would cause all invocations of `more` to run with `-c` set. Normally, the user will place the command sequences which sets up the `MORE` environment variable in the `.cshrc` or `.profile` file.

If `more` is reading from a file, rather than a pipe, then a percentage is displayed along with the **--More--** prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when `more` pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1):

[*i*]`<space>`	Display *i* more lines (or another screenful if no argument is given).
[*i*]`control-d`	Display 11 more lines. If *i* is given, then the scroll size is set to *i*.
[*i*]`d`	Same as control-d.
[*i*]`z`	Same as typing <space> except that *i*, if entered, becomes the new window size.
[*i*]`s`	Skip *i* lines and print a screenful of lines.
[*i*]`f`	Skip *i* screenfuls and print a screenful of lines.
[*i*]`b`	Skip back *i* screenfuls and print a screenful of lines.
[*i*]`control-B`	Same as `b`.
`q` or `Q`	Exit from `more`.
`=`	Display the current line number.
`v`	Start up the editor `vi` at the current line.
`h`	Help command. Gives a description of all the `more` commands.
[*i*]`/`*expr*	Searches for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged, Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression.
[*i*]`n`	Search for the *i*-th occurrence of the last regular expression entered.
‘	(single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*	Invoke a shell with *command*. The characters `%` and `!` in *command* are replaced with the current file name and the previous shell command respectively. If there is no current file name, `%` is not expanded. The sequences `\%` and `\!` are replaced by `%` and `!`, respectively.
[*i*]:n	Skip to the *i*-th next file given in the command line. (Skips to the last file if *i* doesn't make sense.)
[*i*]:p	Skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then `more` goes back to the beginning of the file. If *i* does not make sense, `more` skips back to the first file. If `more` is not reading from a file, the bell is rung and nothing else happens.
:f	Display the current file name and line number.
:q or :Q	Exit from `more` (same as `q` or `Q`).
. (dot)	Repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being typed. In addition, the user may hit the erase character to redisplay the **`--More--(xx%)`** prompt.

When output is being sent to the terminal, the user can type the quit character. `more` will stop sending output, and will display the usual **`--More--`** prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when quit signal occurs.

The terminal is set to `noecho` mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the `/` and `!` commands.

If the standard output is not a teletype, then `more` acts just like `cat`, except that a header is printed before each file (if there is more than one).

If you do not know what the line kill or quit characters for your terminal are, you can find this information by using the `stty` command.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

FILES

`/etc/termcap`	Terminal data base
`/usr/lib/more.help`	Help file

SEE ALSO

Utility Programs - `csh, sh, stty`

NAME

mount – mount a file system or a remote NFS directory

SYNOPSIS

mount [*device_name dir* **[-r]]**

mount -a [-q]

mount -a -[fvq] [-t *type***]**

mount [*hostname***:***remdir dir* **]**

mount [-fvq] [-o *options***]** *filesystem directory*

DESCRIPTION

mount makes the system associate a removable file system to the existing file system hierarchy under a specified directory *dir*. This directory must already exist and does not have to be empty. However, the contents of a non-empty directory that is a mount target will remain hidden until the file system is unmounted.

If the file system is of the form hostname:remdir, it is assumed to be an NFS-type file system. Otherwise, mount will interrogate the file system and determine if it is a Lynx FLFS or an ISO9660 file system.

Both mount and umount maintain the list of currently mounted file system details in the file /etc/mtab. Invoked without any arguments, mount displays the currently mounted file systems by referring to this file. Optionally, mount refers to the file /etc/fstab to mount multiple file systems simultaneously.

OPTIONS

-a Attempt to mount all file systems described in /etc/fstab except the root file system.

-r Mount the file system read-only.

-q Mount the filesystems without making an entry in /etc/mtab.

-f	Fake an /etc/mtab entry, without actually mounting.
-v	Turn on the verbose mode. Display messages as each file system is mounted.
-t *type*	Mount the specified file system *type*. The type could be either block or NFS. The matching type file system entries from /etc/fstab will be mounted.
-o *options*	Specify mount options as either ro or rw to mount the file system with read-only or read/write access. This is the only option that is common for block and NFS type mounts.

OPTIONS SPECIFIC TO NFS MOUNTS

retry=*n*	The number of times to retry the mount operations, in case of soft mounting.
rsize=*n*	The read buffer size is set to *n*.
wsize=*n*	The write buffer size is set to *n*.
timeo=*n*	Set the NFS timeout to *n* tenths of a second (*n**100 ms).
retrans=*n*	The number of NFS retransmissions.
port=*n*	The server IP port number.
soft/hard	Return an error if the server fails to respond or continue the retry request until the server responds.
intr	Allow keyboard interrupts on hard mounts.
acregmin=*n*	Hold cached attributes for at least *n* seconds after modifications.
acregmax=*n*	Hold cached attributes for at most *n* seconds after file modifications.
acdirmin=*n*	Hold cached attributes for at least *n* seconds after directory update.
acdirmax=*n*	Hold cached attributes for at most *n* seconds after directory update.

`actimeo=`*n*	Set minimum and maximum times for regular files and directories to *n* seconds.
`nodc`	Suppress data caching.

NOTE

The default values used for NFS mounting are `fg`, `retry=1`, `timeo=7`, `retrans=3`, `port=NFS_PORT`, `hard`, `acregmin=10`, `acregmax=60`, `acdirmin=30`, `acdirmax=60`, `rsize=4096`, and `wsize=4096`.

EXAMPLES

The following examples show the functions of `mount` with different options when called from the command line shell prompt.

To display the currently mounted file systems:

```
mount
```

To mount all entries in `/etc/fstab`:

```
mount -a
```

To mount all entries from `/etc/fstab`, and do not update `/etc/mtab`:

```
mount -a -q
```

To mount all block device entries in `/etc/fstab`:

```
mount -a -v -t block
```

To mount the block device `sd1542` on `/mnt` directory:

```
mount /dev/sd1542.0a /mnt
```

To mount the block device `sd1542` in read-only mode on `/mnt` directory:

```
mount /dev/sd1542.0a /mnt -r
```

or

mount -o ro /dev/sd1542.0a /mnt

To mount all NFS entries from `/etc/fstab`:

mount -a -v -t NFS

To NFS mount the directory `/usr/dl` of topcat on `/mnt` and use the default caching options

mount topcat:/usr/dl /mnt

To NFS mount the directory `/usr/dl` of topcat on `/mnt` and do not enable caching:

mount -o nocache topcat:/usr/dl /mnt

To NFS mount the directory `/usr/dl` of topcat on `/mnt` with the options `No Data Cache`, `NFS Read Size=8192`, and `NFS Write Size = 8192`:

mount -o nodc,rsize=0x2000,wsize=0x2000 topcat:/usr/dl /mnt

Both mount and umount maintain a table of mounted devices in `/etc/mtab`. This table is displayed if mount is invoked without arguments.

DIAGNOSTICS

Physically write-protected file systems (like magnetic tapes) must be mounted read-only or access errors will result when access times of files on them are updated. Also, if a file system that is full of garbage is mounted, it can crash the system.

FILES

`/etc/mtab`	Mount table.
`/etc/fstab`	File system table.

SEE ALSO

Utility Programs - `umount`

System Calls - `mount()`, `umount()`

NAME

mountd – NFS mount request server

SYNOPSIS

mountd

DESCRIPTION

mountd is an RPC server that answers NFS file system mount requests. It reads the file /etc/xtab, described in exports, to determine which file systems are available for mounting by which machines. It also provides information as to what file systems are mounted by which clients. This information can be printed using the showmount command.

The mountd daemon is normally invoked by /net/rc.network.

FILES

/etc/xtab

SEE ALSO

Utility Programs - showmount

File Formats - exports

NAME

mt – magnetic tape I/O

SYNOPSIS

mt [**–f raw tape device name**] **options** [**count**]

DESCRIPTION

mt controls magnetic tape I/O by sending ioctl commands to the magnetic tape drivers. If the raw tape device name is not specified using the –f option, the environment variable TAPE is used. If TAPE does not exist, the default device /dev/nrtape is used. The same command may be performed multiple number of times by specifying **count**, if **count** is not specified, the requested command is performed once.

mt returns exit status:

0:	operation(s) were successful
1:	unrecognizable command or failure to open tape device
2:	operation failed.

OPTIONS

eof, weof	Write **count** filemarks at current position on tape

FILES

/dev/nrtape	SCSI no-rewind tape interface.

NAME

`mv` – move or rename files

SYNOPSIS

`mv [-fi] [-]` *file1 file2*

`mv [-fi] [-]` *file ... directory*

DESCRIPTION

In the first form, `mv` changes the name of *file1* to *file2*. If pathname *file2* describes a different directory than *file1*, then the link in the original directory is removed and a link is created in the new directory. In the second form, each *file* is moved from its former directory into *directory*, retaining its base name. In either form, if the move crosses a file system boundary, `mv` copies the information from the old file to the new file, then removes the old file.

If the destination file exists, and its permissions disallow writing, `mv` prompts the user and completes the move only on a response of `y`.

OPTIONS

`-` Allows first filename to begin with "**`-`**".

`-f` Force writing over existing files. Overrides the **`-i`** switch.

`-i` The user is prompted whenever a destination exists, regardless of permissions. The move is completed only on response `y`.

NOTE

`mv` does not allow a file to be moved onto itself. The side-effects of cross-file system moves are that the user ID of the copying process becomes the owner of the new file, and all linking information tied to other files in the original file system is lost.

SEE ALSO

Utility Programs -	`cp, ln`
System Calls -	`rename()`

NAME

`named` – Internet domain name server

SYNOPSIS

named [-d *debuglevel* **] [-p** *port#* **] [-b** *bootfile* **]**

DESCRIPTION

`named` is the Internet domain name server. (See RFC883 for more information on the Internet name-domain system.) Without any arguments, `named` will read the default boot file `/etc/named.boot`, read any initial data, and listen for queries. Options are:

-d Print debugging information. A number after the **d** determines the level of messages printed.

-p Use a different port number. The default is the standard port number as listed in `/etc/services`.

-b Use an alternate boot file. This is optional and allows you to specify a file with a leading dash.

Any additional argument is taken as the name of the boot file. The `boot` file contains information about where the name server is to get its initial data. If multiple boot files are specified, only the last is used. Lines in the boot file cannot be continued on subsequent lines. The following is a small example:

```
;
; boot file for name server
directory /usr/local/domain
;
; type domain source host/file
;
domain berkeley.edu
primary Berkeley.edu
secondary cc.berkeley.edu 10.2.0.78 128.32.0.10
cache named.ca
```

The "directory" line causes the server to change its working directory to the directory specified. This can be important for the correct processing of `$INCLUDE` files in primary zone files.

The "cache" line specifies that data in `root.cache` is to be placed in the backup cache. Its main use is to specify data such as locations of root domain servers. This cache is not used during normal operation, but is used as "hints" to find the current root servers. The file `root.cache` is in the same format as `berkeley.edu.zone`. There can be more than one "cache" file specified. The cache files are processed in such a way as to preserve the time-to-live's of data dumped out. Data for the root name servers is kept artificially valid if necessary.

The first "primary" line states that the file `berkeley.edu.zone` contains authoritative data for the `Berkeley.EDU` zone. The file `berkeley.edu.zone` contains data in the master file format described in RFC883. All domain names are relative to the origin, in this case, `Berkeley.EDU` (see below for a more detailed description). The second `primary` line states that the file `ucbhosts.rev` contains authoritative data for the domain `32.128.IN-ADDR.ARPA`, which is used to translate addresses in network `128.32` to hostnames. Each master file should begin with an `SOA` record for the zone (see below).

The first `secondary` line specifies that all authoritative data under `CC.Berkeley.EDU` is to be transferred from the name server at `128.32.137.8`. If the transfer fails it will try 128.32.137.3 and continue trying the addresses, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The first non-dotted-quad address on this line will be taken as a file name in which to backup the transferred zone.

The name server will load the zone from this backup file if it exists when it boots, providing a complete copy even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file will be updated. The second `secondary` line states that the address-to-hostname mapping for the subnet `128.32.136` should be obtained from the same list of master servers as the previous zone.

The `forwarders` line specifies the addresses of site-wide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, then the server will send all queries for data not in the cache to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted.

If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarder's line unless it is in `slave` mode. The forwarding facility is useful to cause a large site-wide cache to be generated on a master, and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do.

The `slave` line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option is normally used on machines that wish to run a server but for physical or administrative reasons cannot be given access to the Internet, but have access to a host that does have access.

The `sortlist` line can be used to indicate networks that are to be preferred over other, unlisted networks. Queries for host addresses from hosts on the same network as the server will receive responses with local network addresses listed first, then addresses on the sort list, then other addresses. This line is only acted on at initial startup. When reloading the nameserver with a `SIGHUP`, this line will be ignored.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

> `$INCLUDE` <*filename*> <*opt_domain*>
> `$ORIGIN` <*domain*> <*domain*> <*opt_ttl*> <*opt_class*> <*type*>
> <*resource_record_data*>

where *domain* is "`.`" for root, "`@`" for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with "`.`", the current origin is appended to the domain. Domain names ending with "`.`" are unmodified.

The *opt_domain* field is used to define an origin for the data in an included file. It is equivalent to placing a `$ORIGIN` statement before the first line of the included file. The field is optional. Neither the *opt_domain* field nor `$ORIGIN` statements in the included file modify the current origin for this file. The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the `SOA` record for the zone.

The *opt_class* field is the object address type; currently only one type is supported, (`IN`), for objects connected to the DARPA Internet. The `type` field contains one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

`A`	A host address (dotted quad).
`NS`	An authoritative name server (domain).
`MX`	A mail exchanger (domain).
`CNAME`	The canonical name for an alias (domain).
`SOA`	Marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL (see RFC883)).
`MB`	A mailbox domain name (domain).
`MG`	A mail group member (domain).
`MR`	A mail rename domain name (domain).
`NULL`	A null resource record (no format or data).
`WKS`	A well known service description (not implemented yet).
`PTR`	A domain name pointer (domain).
`HINFO`	Host information (`cpu_type OS_type`).
`MINFO`	Mailbox or mail list information (`request_domain error_domain`).

Resource records normally end at the end of a line, but may be continued across lines between opening and closing parentheses. Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```
@ IN SOA ucbvax.Berkeley.EDU. rwh.ucbvax.Berkeley.EDU. (
  2.89 ; serial
  10800 ; refresh
  3600 ; retry
  3600000 ; expire
  86400 ) ; minimum
```

The SOA lists a serial number, which should be changed each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds; if the serial number changes, a zone transfer will be done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

NOTE

The boot file directives “domain” and “suffixes” have been obsoleted by a more useful resolver-based implementation of suffixing for partially qualified domain names. The prior mechanisms could fail under a number of situations, especially when then local nameserver did not have complete information.

The following signals have the specified effect when sent to the server process using the `kill` command:

SIGHUP	Causes server to read `named.boot` and reload database.
SIGINT	Dumps current data base and cache to `/usr/tmp/named_dump.db.`
SIGQUIT	Causes the server to checkpoint the cache into the first “cache” file.
SIGIOT	Dumps statistics data into `/usr/tmp/named.stats` if the server is compiled `-DSTATS`. Statistics data is appended to the file.

`SIGSYS`	Dumps the profiling data in `/usr/tmp` if the server is compiled with profiling (server forks, chdirs and exits).
`SIGTERM`	Dumps the primary and secondary database files. Used to save modified data on shutdown if the server is compiled with dynamic updating enabled.
`SIGUSR1`	Turns on debugging; each `SIGUSR1` increments debug level. (`SIGEMT` on older systems without `SIGUSR1`.)
`SIGUSR2`	Turns off debugging completely. (`SIGFPE` on older systems without `SIGUSR2`.)

FILES

`/etc/named.boot`	Name server configuration boot file.
`/etc/named.pid`	The process ID.
`/usr/tmp/named.run`	Debug output.
`/usr/tmp/named_dump.db`	Dump of the name server database.
`/usr/tmp/named.stats`	Nameserver statistics data.

ACKNOWLEDGEMENTS

This section was developed by the University of California, Berkeley.

SEE ALSO

Utility Programs - `kill, hostname`

System Calls - `gethostname(), kill(), sigvec()`

Library Functions - `gethostent(), signal()`

NAME

`netstat` – show network status

SYNOPSIS

netstat [–Aan] [–f *address_family***] [–M** *core***] [–N** *system***]**

netstat [–dghimnrs] [–f *address_family***] [–M** *core***] [–N** *system***]**

netstat [–dn] [–I *interface***] [–M** *core***] [–N** *system***] [–w** *wait***]**

netstat [–p *protocol***] [–I** *interface***] [–M** *core***] [–N** *system***]**

DESCRIPTION

The `netstat` command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with a *wait* interval specified, `netstat` will continuously display the information regarding packet traffic on the configured network interfaces. The fourth form displays statistics about the named protocol.

OPTIONS

–A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

–a With the default display, show the state of all sockets; normally sockets used by server processes are not shown.

–d With either interface display (option **–i** or an interval, as described below), show the number of dropped packets.

–f *address_family*
Limit statistics or address control block reports to those of the specified address family. The following address families are recognized: `inet` for AF_INET.

-g Show information related to multicast (group address) routing. By default, show the IP Multicast virtual-interface and routing tables. If the **-s** option is also present, show multicast routing statistics.

-h Show the state of the IMP host table (obsolete).

-I *interface* Show information about the specified interface; used with a wait interval as described below.

-i Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown). If the **-a** options is also present, multicast addresses currently in use are shown for each Ethernet interface and for each IP interface address. Multicast addresses are shown on separate lines following the interface address with which they are associated.

-M Extract values associated with the name list from the specified core instead of the default `/dev/kmem`.

The **-M** option is not applicable to the LynxOS implementation.

-m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).

The **-m** option is not applicable to the LynxOS implementation.

-N Extract the name list from the specified system instead of the default `/vmunix`.

-n Show network addresses as numbers (normally `netstat` interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.

-p *protocol* Show statistics about *protocol*, which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the file `/etc/protocols`. A null response typically means that there are no interesting numbers to report. The program will complain if *protocol* is unknown or if there is no statistics routine for it.

-s Show per-protocol statistics. If this option is repeated, counters with a value of zero are suppressed.

-r Show the routing tables. When **-s** is also present, show routing statistics instead.

-w *wait* Show network interface statistics at intervals of wait seconds.

USAGE

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form "*host*.*port*" or "*network*.*port*" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases `/etc/hosts` and `/etc/networks`, respectively. If a symbolic name for an address is unknown, or if the **-n** option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet "dot format," refer to `inet_addr`(3)). Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows a collection of information about the route stored as binary choices. The individual flags are discussed in more detail in `route`(1). The mapping between letters and flags is:

```
1   RTF_PROTO2    Protocol specific routing flag #1
2   RTF_PROTO1    Protocol specific routing flag #2
B   RTF_BLACKHOLE Just discard pkts (during updates)
C   RTF_CLONING   Generate new routes on use
D   RTF_DYNAMIC   Created dynamically (by redirect)
G   RTF_GATEWAY   Destination requires forwarding by intermediary
H   RTF_HOST      Host entry (net otherwise)
L   RTF_LLINFO    Valid protocol to link address translation.
M   RTF_MODIFIED  Modified dynamically (by redirect)
R   RTF_REJECT    Host or net unreachable
S   RTF_STATIC    Manually added
U   RTF_UP        Route usable
X   F_XRESOLVE    External daemon translates proto to link address
```

Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When `netstat` is invoked with the **-w** option and a *wait* interval argument, it displays a running count of statistics related to network interfaces. An obsolescent version of this option used a numeric parameter with no option, and is currently supported for backward compatibility. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the **-I** option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

Utility Programs - `nfsstat`, `ps`

FIle Formats - `hosts`, `networks`, `protocols`, `services`

HISTORY

The `netstat` command appeared in 4.2BSD.

BUGS

The notion of errors is ill-defined.

NAME

newconsole – choose a new console device

SYNOPSIS

newconsole [*dev_name***]**

DESCRIPTION

newconsole lets the user choose a new console other than the default console device as initialized by the LynxOS at boot time. newconsole initializes the console to the device given by the argument *dev_name*. If no argument is given, stdout is used as default console.

Also note, only root can change the console device.

DIAGNOSTICS

newconsole fails on account of the following:

If the *dev_name* is not a character device.

If the user issuing the command is not a superuser.

SEE ALSO

System Calls - newconsole()

NAME

nfsd – NFS daemon

SYNOPSIS

nfsd [*nservers* **]**

DESCRIPTION

nfsd starts the daemons that handle client file system requests. nservers is the number of file system request daemons to start. This number should be based on the load expected on this server. Four seems to be a good number.

When a file that is opened by a client is unlinked (by the server), a file with a name of the form .nfsXXX (where XXX is a number) is created by the client. When the open file is closed, the .nfsXXX file is removed. If the client crashes before the file can be closed, the .nfsXXX file is not removed.

FILES

.nfsXXX	client machine pointer to an open-but-unlinked file

SEE ALSO

Utility Programs - mountd

File Formats - exports

NAME

`nfsstat` – Network File System Statistics

SYNOPSIS

`nfsstat [–csz]`

DESCRIPTION

`nfsstat` displays statistical information about the NFS (Network File System) operations in the kernel. It can also be used to reinitialize this information. If no options are give, the default is:

```
nfsstat –cs
```

The display shows the counts for the various NFS protocol requests. For the client side it shows the count for each NFS request type sent over the network. If the NFS cache device driver module is configured, it also displays the count for each type of request made to the NFS cache device driver in the kernel. The difference between these counts is an indication of the efficiency of the NFS caching.

OPTIONS

–c Display Client information

–s Display Server information

–z Zero the specified statistics. This option is for the Superuser only. It can be combined with either -c or -s options to zero that particular sets of statistics.

NAME

`nice` – run a command at a lower priority

SYNOPSIS

`nice [[-]` *number***`]`** *command* **`[`***arguments***`]`**

DESCRIPTION

`nice` functions in much the same way as `prio(1)` except that the +/- increment acts in the opposite direction. If *number* is positive, `nice` executes *command* at the calling process' priority minus *number*. If *number* is negative, nice executes *command* at the calling process' priority plus *number*. If *number* is not specified, *command* is executed at the calling process' priority minus 10.

Unlike the `prio` utility, the *command* argument to `nice` is not optional.

EXAMPLES

Execute `ls` at the priority of the calling process + 1:

```
nice -1 ls
```

Execute `ls` at the priority of the calling process - 2:

```
nice 2 ls
```

Execute `ls` at the priority of the calling process - 10:

```
nice ls
```

SEE ALSO

Utility Programs - `prio`

NAME

`nm` – list symbols from object files.

SYNOPSIS

nm [-a|--debug-syms] [-g|--extern-only] [-B] [-C|--demangle] [-D|--dynamic] [-s|--print-armap] [-o|--print-file-name] [-n|--numeric-sort] [-p|--no-sort] [-r|--reverse-sort] [--size-sort] [-u|--undefined-only] [-t *radix***|--radix=***radix***] [-P|--portability] [-f** *format***|--format=***format***] [--target=***bfdname***] [** *objfile***...]**

nm --help

nm --version

DESCRIPTION

GNU `nm` lists the symbols from object files *objfile*. If no object files are given as arguments, `nm` assumes ‘`a.out`’.

OPTIONS

The long and short forms of options, shown here as alternatives, are equivalent.

-A
-o
--print-file-name
: Precede each symbol by the name of the input file where it was found, rather than identifying the input file once only before all of its symbols.

-a
--debug-syms
: Display debugger-only symbols; normally these are not listed.

-B
: The same as `--format=bsd` (for compatibility with the MIPS `nm`).

-C
--demangle Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable.

-D
--dynamic Display the dynamic symbols rather than the normal symbols. This is only meaningful for dynamic objects, such as certain types of shared libraries.

-f *format* Use the output format *format*, which can be "`bsd`", "`sysv`", or "`posix`". The default is "`bsd`". Only the first character of *format* is significant; it can be either upper or lower case.

-g
--extern-only Display only external symbols.

-n
-v
--numeric-sort
Sort symbols numerically by their addresses, not alphabetically by their names.

-p
--no-sort Don't bother to sort the symbols in any order; just print them in the order encountered.

-P
--portability Use the POSIX.2 standard output format instead of the default format. Equivalent to "**-f** posix".

-s
--print-armap When listing symbols from archive members, include the index: a mapping (stored in the archive by `ar` or `ranlib`) of what modules contain definitions for what names.

-r
--reverse-sort
Reverse the sense of the sort (whether numeric or alphabetic); let the last come first.

--size-sort Sort symbols by size. The size is computed as the difference between the value of the symbol and the value of the symbol with the next higher value. The size of the symbol is printed, rather than the value.

-t *radix*
--radix=*radix* Use *radix* as the radix for printing the symbol values. It must be “d” for decimal, “o” for octal, or “x” for hexadecimal.

--target=*bfdname*
Specify an object code format other than your system’s default format. See `objdump(1)`, for information on listing available formats.

-u
--undefined-only
Display only undefined symbols (those external to each object file).

-V
--version Show the version number of `nm` and exit.

--help Show a summary of the options to `nm` and exit.

SEE ALSO

Utility Programs - `ar, objdump, ranlib`

Documents - ‘`binutils`’ entry in `info`;

The GNU Binary Utilities, Roland H. Pesch

COPYING

NAME

`nohup` – run a command immune to hang-up

SYNOPSIS

`nohup` *command-line*

`nohup --help`

`nohup --version`

DESCRIPTION

This manual page documents the GNU version of `nohup`. `nohup` runs the given command with hang-up signals ignored and the scheduling priority increased by 5, so that the command can continue running in the background after you log out.

If the standard output is a tty, it and the standard error are redirected so that they are appended to the file `nohup.out`; if that cannot be written to, they are appended to the file `$HOME/nohup.out`. If that cannot be written to, the command is not run.

If `nohup` creates the file `nohup.out` or `$HOME/nohup.out`, it creates it with no access permissions for group or other. It does not change the permissions of those files if they already existed. `nohup` does not automatically put the command it runs in the background; you must do that explicitly, by ending the command line with an `&`.

OPTIONS

When nohup is invoked with exactly one argument, the following options are recognized.

`--help` Print a usage message on standard output and exit successfully.

`--version` Print version information on standard output then exit successfully.

NAME

`ntpdate` – set the date and time via NTP

SYNOPSIS

`ntpdate [-bdos] [-a key#] [-e authdelay] [-k keyfile]`
`[-o version] [-p samples] [-t timeout ] server ...`

DESCRIPTION

`ntpdate` sets the local date and time by polling the Network Time Protocol server(s) on the host(s) given as arguments to determine the correct time. It must be run as root on the local host. A number of samples are obtained from each of the servers specified and the standard NTP clock filter and selection algorithms are applied to select the best of these. Typically, `ntpdate` can be inserted in the `/bin/rc` or `/net/rc.network` startup script to set the time of day at boot time and/or can be run from time-to-time via `cron`. Note that `ntpdate`'s reliability and precision will improve dramatically with greater numbers of servers. While a single server may be used, better performance and greater resistance to insanity on the part of any one server will be obtained by providing at least three or four servers, if not more.

Time adjustments are made by `ntpdate` in one of two ways. If `ntpdate` determines your clock is off by more than 0.5 seconds it will simply step the time by calling `settimeofday`. If the error is less than 0.5 seconds, however, it will by default slew the clock's time via a call to `adjtime` with the offset. The latter technique is less disruptive and more accurate when the offset is small, and works quite well when `ntpdate` is run by `cron` every hour or two. The adjustment made in the latter case is actually 50% larger than the measured offset since this will tend to keep a badly drifting clock more accurate (at some expense to stability, though this trade-off is usually advantageous). At boot time, however, it is usually better to always step the time. This can be forced in all cases by specifying the **`-b`** switch on the command line. The **`-s`** switch tells `ntpdate` to log its actions via the `syslog` facility rather than to the standard output, a useful option when running the program from `cron`.

The -d flag may be used to determine what `ntpdate` will do without it actually doing it. Information useful for general debugging will also be printed. By default `ntpdate` claims to be an NTP version 3 implementation in its outgoing packets. As some older software will decline to respond to version 2 queries, the **-o** switch can be used to force the program to poll as a version 1 or version 2 implementation instead.

The number of samples `ntpdate` acquires from each server can be set to between 1 and 8 inclusive using the -p switch.The default is 4. The time it will spend waiting for a response can be set using the **-t** switch, and will be rounded to a multiple of 0.2 seconds. The default is 1 second, a value suitable for polling across a LAN.

`ntpdate` will authenticate its transactions if need be.The **-a** switch specifies that all packets should be authenticated using the key number indicated. The **-k** switch allows the name of the file from which the keys may be read to be modified from the default of `/etc/ntp.keys`. This file should be in the format described in `xntpd`. The **-e** option allows the specification of an authentication processing delay, in seconds (see `xntpd` manual page for details). This number is usually small enough to be negligible for `ntpdate`'s purposes, though specifying a value may improve timekeeping on very slow CPU's.

The -u options tells `ntpdate` to use an unprivileged port to send the packets from. This is most useful when you are behind a firewall that blocks incoming traffic to privileged ports, and you want to synchronize with hosts beyond the firewall. Note that the -d option always uses unprivileged ports.

`ntpdate` will decline to set the date if an NTP server daemon(e.g. `xntpd`) is running on the same host. When running `ntpdate` on a regular basis from `cron` as an alternative to running a daemon, doing so once every hour or two will result in precise enough time-keeping to avoid stepping the clock.

FILES

`/etc/ntp.keys`	Contains the encryption keys used by ntpdate.

SEE ALSO

Utility Programs - `xntpd`

HISTORY

Written by Dennis Ferguson at the University of Toronto.

COPYRIGHT NOTICE

NAME

`ntpq` – standard Network Time Protocol query program

SYNOPSIS

ntpq [-inp] [-c *command***] [***host***] [...]**

DESCRIPTION

`ntpq` is used to query NTP servers which implement the recommended NTP mode 6 control message format about current state and to request changes in that state. The program may be run either in interactive mode or controlled using command line arguments. Requests to read and write arbitrary variables can be assembled, with raw and pretty-printed output options being available. `ntpq` can also obtain and print a list of peers in a common format by sending multiple queries to the server.

If one or more request options is included on the command line when `ntpq` is executed, each of the requests will be sent to the NTP servers running on each of the hosts given as command line arguments, or on localhost by default. If no request options are given, `ntpq` will attempt to read commands from the standard input and execute these on the NTP server running on the first host given on the command line, again defaulting to localhost when no other host is specified. `ntpq` will prompt for commands if the standard input is a terminal device.

`ntpq` uses NTP mode 6 packets to communicate with the NTP server, and hence can be used to query any compatible server on the network which permits it. Note that since NTP is a UDP protocol this communication will be somewhat unreliable, especially over large distances in terms of network topology. `ntpq` makes one attempt to retransmit requests, and will time requests out if the remote host is not heard from within a suitable time out time.

Command line options are described following. Specifying a command line option other than **-i** or **-n** will cause the specified query (queries) to be sent to the indicated host(s) immediately. Otherwise, `ntpq` will attempt to read interactive format commands from the standard input.

-c	The following argument is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). Multiple -c options may be given.
-i	Force `ntpq` to operate in interactive mode. Prompts will be written to the standard output and commands read from the standard input.
-n	Output all host addresses in dotted-quad numeric format rather than converting to the canonical host names.
-p	Print a list of the peers known to the server as well as a summary of their state. This is equivalent to the "peers" interactive command.

INTERNAL COMMANDS

Interactive format commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output of a command is normally sent to the standard output, but optionally the output of individual commands may be sent to a file by appending a ">", followed by a file name, to the command line.

A number of interactive format commands are executed entirely within the ntpq program itself and do not result in NTP mode 6 requests being sent to a server. These are described following.

```
? [ command_keyword }
```

A “?” by itself will print a list of all the command keywords known to this incarnation of ntpq. A “?” followed by a command keyword will print function and usage information about the command. This command is probably a better source of information about ntpq than this manual page.

```
timeout millseconds
```

Specify a time out period for responses to server queries. The default is about 5000 milliseconds. Note that since ntpq retries each query once after a time out the total waiting time for a time out will be twice the time out value set.

```
delay milliseconds
```

Specify a time interval to be added to timestamps included in requests which require authentication. This is used to enable (unreliable) server reconfiguration over long delay network paths or between machines whose clocks are unsynchronized. Actually the server does not now require time stamps in authenticated requests, so this command may be obsolete.

```
host hostname
```

Set the host to which future queries will be sent. Hostname may be either a host name or a numeric address.

```
poll [ # ] [ verbose ]
```

Poll the current server in client mode. The first argument is the number of times to poll (default is 1) while the second argument may be given to obtain a more detailed output of the results. This command is currently just wishful thinking.

```
keyid #
```

This command allows the specification of a key number to be used to authenticate configuration requests. This must correspond to a key number the server has been configured to use for this purpose.

```
passwd
```

This command prompts you to type in a password (which will not be echoed) which will be used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose if such requests are to be successful.

```
hostnames yes|no
```

If “yes” is specified, host names are printed in information displays. If “no” is given, numeric addresses are printed instead. The default is “yes” unless modified using the command line -n switch.

```
raw
```

Causes all output from query commands is printed as received from the remote server. The only formatting/interpretation done on the data is to transform non-ASCII data into a printable (but barely understandable) form.

`cooked`

Causes output from query commands to be “cooked”. Variables which are recognized by the server will have their values reformatted for human consumption. Variables which `ntpq` thinks should have a decodeable value but didn’t are marked with a trailing “?”.

`ntpversion 1|2|3`

Sets the NTP version number which `ntpq` claims in packets. Defaults to 3, Note that mode 6 control messages (and modes, for that matter) didn’t exist in NTP version 1. There appear to be no servers left which demand version 1.

`authenticate yes|no`

Normally `ntpq` does not authenticate requests unless they are write requests. The command authenticate yes causes `ntpq` to send authentication with all requests it makes. Authenticated requests causes some servers to handle requests slightly differently, and can occasionally melt the CPU in fuzzballs if you turn authentication on before doing a peer display.

`addvars` *variable_name*`[=`*value*`][,...]`
`rmvars` *variable_name*`[,...]` `clearvars`

The data carried by NTP mode 6 messages consists of a list of items of the form

variable_name=value

where the “=*value*” is ignored, and can be omitted, in requests to the server to read variables. `ntpq` maintains an internal list in which data to be included in control messages can be assembled, and sent using the `readlist` and `writelist` commands described below. The `addvars` command allows variables and their optional values to be added to the list. If more than one variable is to be added, the list should be comma-separated and not contain white space. The `rmvars` command can be used to remove individual variables from the list, while the `clearlist` command removes all variables from the list.

```
debug more|less|off
```

Turns internal query program debugging on and off.

```
quit
```

Exit `ntpq`.

CONTROL MESSAGE COMMANDS

Each peer known to an NTP server has a 16 bit integer association identifier assigned to it. NTP control messages which carry peer variables must identify the peer the values correspond to by including its association ID. An association ID of 0 is special, and indicates the variables are system variables, whose names are drawn from a separate name space.

Control message commands result in one or more NTP mode 6 messages being sent to the server, and cause the data returned to be printed in some format. Most commands currently implemented send a single message and expect a single response. The current exceptions are the peers command, which will send a pre-programmed series of messages to obtain the data it needs, and the `mreadlist` and `mreadvar` commands, which will iterate over a range of associations.

```
associations
```

Obtains and prints a list of association identifiers and peer statuses for in-spec peers of the server being queried. The list is printed in columns. The first of these is an index numbering the associations from 1 for internal use, the second the actual association identifier returned by the server and the third the status word for the peer. This is followed by a number of columns containing data decoded from the status word. Note that the data returned by the `associations` command is cached internally in `ntpq`. The index is then of use when dealing with stupid servers which use association identifiers which are hard for humans to type, in that for any subsequent commands which require an association identifier as an argument, the form &index may be used as an alternative.

`lassocations`

Obtains and prints a list of association identifiers and peer statuses for all associations for which the server is maintaining state. This command differs from the `associations` command only for servers which retain state for out-of-spec client associations (i.e. fuzzballs). Such associations are normally omitted from the display when the `associations` command is used, but are included in the output of `lassociations`.

`passociations`

Prints association data concerning in-spec peers from the internally cached list of associations. This command performs identically to the `associations` except that it displays the internally stored data rather than making a new query.

`lpassociations`

Print data for all associations, including out-of-spec client associations, from the internally cached list of associations. This command differs from `passociations` only when dealing with fuzzballs.

`pstatus assocID`

Sends a read status request to the server for the given association. The names and values of the peer variables returned will be printed. Note that the status word from the header is displayed preceding the variables, both in hexadecimal and in pidgeon English.

`readvar [ assocID ] [` *variable_name*`[=`*value*`][,...] ]`

Requests that the values of the specified variables be returned by the server by sending a read variables request. If the association ID is omitted or is given as zero the variables are system variables, otherwise they are peer variables and the values returned will be those of the corresponding peer. Omitting the variable list will send a request with no data which should induce the server to return a default display.

`rv [ assocID ] [` *variable_name*`[=`*value*`][,...] ]`

An easy-to-type short form for the `readvar` command.

`writevar assocID` *variable_name*=*value*`[,...]`

Like the readvar request, except the specified variables are written instead of read.

```
readlist [ assocID ]
```

Requests that the values of the variables in the internal variable list be returned by the server. If the association ID is omitted or is 0 the variables are assumed to be system variables. Otherwise they are treated as peer variables. If the internal variable list is empty a request is sent without data, which should induce the remote server to return a default display.

```
rl [ assocID ]
```

An easy-to-type short form of the `readlist` command.

```
writelist [ assocID ]
```

Like the `readlist` request, except the internal list variables are written instead of read.

`mreadvar assocID assocID [` *variable_name*`[=`*value*`][,...] ]`

Like the `readvar` command except the query is done for each of a range of (nonzero) association IDs. This range is determined from the association list cached by the most recent associations command.

`mrv assocID assocID [` *variable_name*`[=`*value*`][,...] ]`

An easy-to-type short form of the `mreadvar` command.

```
mreadlist assocID assocID
```

Like the `readlist` command except the query is done for each of a range of (nonzero) association IDs. This range is determined from the association list cached by the most recent associations command.

```
mrl assocID assocID
```

An easy-to-type short form of the `mreadlist` command.

clockvar [assocID] [*variable_name*[=*value*][,...]]

Requests that a list of the server's clock variables be sent. Servers which have a radio clock or other external synchronization will respond positively to this. If the association identifier is omitted or zero the request is for the variables of the "system clock" and will generally get a positive response from all servers with a clock. If the server treats clocks as pseudo-peers, and hence can possibly have more than one clock connected at once, referencing the appropriate peer association ID will show the variables of a particular clock. Omitting the variable list will cause the server to return a default variable display.

cv [assocID] [*variable_name*[=*value*][,...]]

An easy-to-type short form of the `clockvar` command.

```
peers
```

Obtains a list of in-spec peers of the server, along with a summary of each peer's state. Summary information includes the address of the remote peer, the reference ID (0.0.0.0 if the refID is unknown), the stratum of the remote peer, the type of the peer (local, unicast, multicast or broadcast), when the last packet was received, the polling interval, in seconds, the reachability register, in octal, and the current estimated delay, offset and dispersion of the peer, all in seconds.

The character in the left margin indicates the fate of this peer in the clock selection process. The codes mean: *sp* discarded due to high stratum and/or failed sanity checks; "x" designated falsticker by the intersection algorithm; "." culled from the end of the candidate list; "-" discarded by the clustering algorithm; "+" included in the final selection set; "#" selected for synchronization but distance exceeds maximum; "*" selected for synchronization; and "o" selected for synchronization, pps signal in use.

Note that since the peers command depends on the ability to parse the values in the responses it gets it may fail to work from time to time with servers which poorly control the data formats.

The contents of the host field may be one of four forms. It may be a host name, an IP address, a reference clock implementation name with its parameter or `REFCLK(<implementation number>, <parameter>)`. On "hostnames no" only IP-addresses will be displayed.

`lpeers`

Like `peers`, except a summary of all associations for which the server is maintaining state is printed. This can produce a much longer list of peers from fuzzball servers.

`opeers`

An old form of the `peers` command with the reference ID replaced by the local interface address.

HISTORY

Written by Dennis Ferguson at the University of Toronto.

BUGS

The `peers` command is non-atomic and may occasionally result in spurious error messages about invalid associations occurring and terminating the command.

The timeout time is a fixed constant, which means you wait a long time for time outs since it assumes sort of a worst case. The program should improve the time out estimate as it sends queries to a particular host, but doesn't.

COPYRIGHT NOTICE

The following copyright notice applies to all files collectively called the Network Time Protocol Version 4 Distribution. Unless specifically declared otherwise in an individual file, this notice applies as if the text was explicitly included in the file.

NAME

`ntptrace` – trace a chain of NTP hosts back to their master time source

SYNOPSIS

ntptrace [-vdn] [-r *retries* **] [-t** *timeout* **] [** *server* **]**

DESCRIPTION

`ntptrace` determines where a given Network Time Protocol (NTP) server gets its time from, and follows the chain of NTP servers back to their master time source. If given no arguments, it starts with localhost.

Here is an example of the output from `ntptrace`:

```
% ntptrace

localhost:stratum 4,offset 0.0019529,synch distance 0.144135
server2.bozo.com:stratum 2,offset 0.0124263,synch distance
0.115784
usndh.edu:stratum 1,offset 0.0019298,synch distance 0.011993,
refid 'WWVB'
```

On each line, the fields are (left to right): the host name, the host's stratum, the time offset between that host and the local host (as measured by `ntptrace`; this is why it is not always zero for localhost), the host's "synchronization distance," and (only for stratum-1 servers) the reference clock ID. All times are given in seconds. (Synchronization distance is a measure of the goodness of the clock's time.)

OPTIONS

-d Turns on some debugging output.

-n Turns off the printing of host names; instead, host IP addresses are given. This may be necessary if a nameserver is down.

-r *retries* Sets the number of retransmission attempts for each host; default = 5.

-t *timeout*	timeout Sets the retransmission timeout (in seconds); default = 2.
-v	Prints verbose information about the NTP servers.

SEE ALSO

Utility Programs- `xntpd, xntpdc`

BUGS

This program makes no attempt to improve accuracy by doing multiple samples.

COPYRIGHT NOTICE

NAME

objcopy – copy and translate object files

SYNOPSIS

objcopy [-F *bfdname* **| --target=***bfdname***]**
[-I *bfdname* **| --input-target=***bfdname***]**
[-O *bfdname* **| --output-target=***bfdname***]**
[-R *sectionname* **| --remove-section=***sectionname***]**
[-S | --strip-all] [-g | --strip-debug] [--strip-unneeded]
[-K *symbolname* **| --keep-symbol=***symbolname***]**
[-N *symbolname* **| --strip-symbol=***symbolname***]**
[-x | --discard-all] [-X | --discard-locals]
[-b *byte* **| --byte=***byte***] [-i** *interleave* **| --interleave=***interleave***]**
[--gap-fill=*val***] [--pad-to=***address***] [--set-start=***val***]**
[--adjust-start=*incr***] [--adjust-vma=***incr***]**
[--adjust-section-vma=*section***{=|+|-}***val***] [--adjust-warnings]**
[--no-adjust-warnings] [--set-section-flags=*section***=***flags***]**
[--add-section=*sectionname***=***filename***] [-v | --verbose]**
infile **[***outfile***]**

objcopy {-V | --version}

objcopy --help

DESCRIPTION

The GNU `objcopy` utility copies the contents of an object file to another. `objcopy` uses the GNU BFD Library to read and write the object files. It can write the destination object file in a format different from that of the source object file. The exact behavior of `objcopy` is controlled by command-line options.

`objcopy` creates temporary files to do its translations and deletes them afterward. `objcopy` uses BFD to do all its translation work; it knows about all the formats BFD knows about, and thus is able to recognize most formats without being told explicitly.

`objcopy` can be used to generate S-records by using an output target of `srec` (e.g., use `-O srec`).

`objcopy` can be used to generate a raw binary file by using an output target of `binary` (e.g., use `-O binary`). When `objcopy` generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file. All symbols and relocation information will be discarded. The memory dump will start at the virtual address of the lowest section copied into the output file.

When generating an S-record or a raw binary file, it may be helpful to use **-S** to remove sections containing debugging information. In some cases **-R** will be useful to remove sections which contain information which is not needed by the binary file.

infile and *outfile* are the source and output files respectively. If you do not specify *outfile*, `objcopy` creates a temporary file and destructively renames the result with the name of the input file.

OPTIONS

-I *bfdname*, **--input-target=***bfdname*
: Consider the source file's object format to be *bfdname*, rather than attempting to deduce it.

-O *bfdname*, **--output-target=***bfdname*
: Write the output file using the object format *bfdname*.

-F *bfdname*, **--target=***bfdname*
: Use *bfdname* as the object format for both the input and the output file; i.e. simply transfer data from source to destination with no translation.

-R *sectionname*, **--remove-section=***sectionname*
: Remove the named section from the file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable.

-S, **--strip-all**
: Do not copy relocation and symbol information from the source file.

-g, **--strip-debug**
: Do not copy debugging symbols from the source file.

--strip-unneeded
Strip all symbols that are not needed for relocation processing.

-K *symbolname*, **--keep-symbol=***symbolname*
Copy only symbol *symbolname* from the source file. This option may be given more than once.

-N *symbolname*, **--strip-symbol=***symbolname*
Do not copy symbol *symbolname* from the source file. This option may be given more than once, and may be combined with strip options other than **-K**.

-x, **--discard-all**
Do not copy non-global symbols from the source file.

-X, **--discard-locals**
Do not copy compiler-generated local symbols. (These usually start with `L` or a period character).

-b *byte*, **--byte=***byte*
Keep only every *byte*th byte of the input file (header data is not affected). *byte* can be in the range from 0 to the interleave-1. This option is useful for creating files to program ROMs. It is typically used with an srec output target.

-i *interleave*, **--interleave=***interleave*
Only copy one out of every *interleave* bytes. Which one to copy is selected by the **-b** or **--byte** option. The default is 4. The interleave is ignored if neither **-b** nor **--byte** is given.

--gap-fill=*val* Fill gaps between sections with *val*. This is done by increasing the size of the section with the lower address, and filling in the extra space created with *val*.

--pad-to=*address* Pad the output file up to the virtual address *address*. This is done by increasing the size of the last section. The extra space is filled in with the value specified by **--gap-fill** (default zero).

--set-start=*val*
Set the start address of the new file to *val*. Not all object file formats support setting the start address.

--adjust-start=*incr*
Adjust the start address by adding *incr*. Not all object file formats support setting the start address.

--adjust-vma=*incr*
Adjust the address of all sections, as well as the start address, by adding *incr*. Some object file formats do not permit section addresses to be changed arbitrarily. Note that this does not relocate the sections; if the program expects sections to be loaded at a certain address, and this option is used to change the sections such that they are loaded at a different address, the program may fail.

--adjust-section-vma=*section*{=|+|-}*val*
Set or adjust the address of the named *section*. If = is used, the section address is set to *val*. Otherwise, *val* is added to or subtracted from the section address. See the comments under **--adjust-vma**, above. If *section* does not exist in the input file, a warning will be issued, unless **--no-adjust-warnings** is used.

--adjust-warnings
If **--adjust-section-vma** is used, and the named section does not exist, issue a warning. This is the default.

--no-adjust-warnings
Do not issue a warning if **--adjust-section-vma** is used, even if the named section does not exist.

--set-section-flags=*section=flags*
Set the flags for the named section. The *flags* argument is a comma separated string of flag names. The recognized names are `alloc`, `load`, `readonly`, `code`, `data`, and `rom`. Not all flags are meaningful for all object file formats.

--add-section=*sectionname=filename*
Add a new section named *sectionname* while copying the file. The contents of the new section are taken from the file *filename*. The size of the section will be the size of the file. This option only works on file formats which can support sections with arbitrary names.

-v, **--verbose**	Verbose output: list all object files modified. In the case of archives, `objcopy -V` lists all members of the archive.
-V, **--version**	Show the version number of `objcopy` and exit.
--help	Show a summary of the options to `objcopy` and exit.

SEE ALSO

Documents - *The GNU Binary Utilities*, Roland H. Pesch.

COPYING

NAME

objdump – display information from object files.

SYNOPSIS

objdump [-a|--archive-headers]
[-b *bfdname* **| --target=***bfdname***]**
[-d|--disassemble] [-D|--disassemble-all]
[-f|--file-headers]
[-h|--section-headers | --headers]
[-i|--info] [-j *section* **| --section=***section***]**
[-l|--line-numbers]
[-m *machine* **| --architecture=***machine***]**
[-r|--reloc] [-R|--dynamic-reloc]
[-s|--full-contents] [-S|--source] [--stabs]
[-t|--syms] [-T|--dynamic-syms]
[-x|--all-headers] [--start-address=*address***]**
[--stop-address=*address***]**
objfile...

objdump --version

objdump --help

DESCRIPTION

objdump displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

objfile files are the object files to be examined. When you specify archives, objdump shows information on each of the member object files.

OPTIONS

Where long and short forms of an option are shown together, they are equivalent. At least one option besides **-l** must be given.

-a
--archive-headers
If any files from *objfile* are archives, display the archive header information (in a format similar to '`ls -l`'). Besides the information you could list with '`ar tv`', '`objdump -a`' shows the object file format of each archive member.

-b *bfdname*
--target=*bfdname*
Specify the object-code format for the object files to be *bfdname*. This may not be necessary; *objdump* can automatically recognize many formats. For example,

```
objdump -b oasys -m vax -h fu.o
```

display summary information from the section headers ('**-h**') of '`fu.o`', which is explicitly identified ('**-m**') as a Vax object file in the format produced by Oasys compilers. You can list the formats available with the '**-i**' option.

-d
--disassemble
Display the assembler mnemonics for the machine instructions from *objfile*. This option only disassembles those sections which are expected to contain instructions.

-D
--disassemble-all
Like **-d**, but disassemble the contents of all sections, not just those expected to contain instructions.

-f
--file-headers
Display summary information from the overall header of each file in *objfile*.

-h
--section-headers
--headers Display summary information from the section headers of the object file.

--help Print a summary of the options to `objdump` and exit.

-i
--info Display a list showing all architectures and object formats available for specification with **-b** or **-m**.

-j *name*
--section=*name*
Display information only for section *name*.

-l
--line-numbers Label the display (using debugging information) with the filename and source line numbers corresponding to the object code shown. Only useful with **-d** or **-D**.

-m *machine*
--architecture=*machine*
Specify the object files *objfile* are for architecture *machine*. You can list available architectures using the '**-i**' option.

-r
--reloc Print the relocation entries of the file. If used with **-d** or **-d**, the relocations are printed interspersed with the disassembly.

-R
--dynamic-reloc
Print the dynamic relocation entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries.

-s
--full-contents
Display the full contents of any sections requested.

-S
--source Display source code intermixed with disassembly, if possible. Implies **-d**.

--stabs Display the contents of the .stab, .stab.index, and `.stab.excl` sections from an ELF file. This is only useful on systems (such as Solaris 2.0) in which `.stab` debugging symbol-table entries are carried in an ELF section. In most other file formats, debugging symbol-table entries are interleaved with linkage symbols, and are visible in the **--syms** output.

--start-address=*address*
Start displaying data at the specified address. This affects the output of the **-d**, **-r** and **-s** options.

--stop-address=*address*
Stop displaying data at the specified address. This affects the output of the **-d**, **-r** and **-s** options.

-t
--syms Symbol Table. Print the symbol table entries of the file. This is similar to the information provided by the 'nm' program.

-T
--dynamic-syms Dynamic Symbol Table. Print the dynamic symbol table entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. This is similar to the information provided by the 'nm' program when given the **-D (--dynamic)** option.

--version Print the version number of `objdump` and exit.

-x
--all-headers Display all available header information, including the symbol table and relocation entries. Using '**-x**' is equivalent to specifying all of '**-a -f -h -r -t**'.

SEE ALSO

Utility Programs - `nm`

Documents - *The GNU Binary Utilities*, Roland H. Pesch.

COPYING

NAME

`od` – file dumping utility (octal, decimal, hex, or ascii)

SYNOPSIS

od [*format***] [***file***] [[+]** *offset* **[.] [bB] [***label***]]**

xd [*format***] [***file***] [[+]** *offset* **[.] [bB] [***label***]]**

DESCRIPTION

`od` displays the contents of the given *file*, or the standard input, in a format specified by its invocation. If no dumping format is specified, the **–o** (octal) format is assumed. Running the `xd` utility is the same as specifying **–x** to `od`.

OPTIONS

The format arguments can be selected from the following:

–a *opt* Display bytes as printable characters. If *opt* is `P` then bytes with odd parity are highlighted; if *opt* is `p` then bytes with even parity are highlighted. Parity is ignored if *opt* is omitted.

–b Display bytes as unsigned octal.

–c Display bytes as printable characters, as they would appear in C strings. Characters which are not displayable are displayed as C escapes (null = `\0`, newline = `\n`, etc.). Other undisplayable characters appear as 3-digit octal integers. Any bytes with the parity bit set are displayed in octal.

–d Display short words (16 bits) as unsigned decimal.

–f Display long words (32 bits) as floating point values.

–h Display short words as unsigned hexadecimal.

–i Display short words as signed decimal.

–l Display long words as signed decimal.

-o	Display short words as unsigned octal.
-s *n*	Look for strings consisting of printable ascii characters terminated by a null byte. The optional parameter *n* specifies the minimum string length to be recognized. The default value for *n* is 3 characters.
-v	Show all data. Without this option, all but the first of a sequence of identical lines are replaced by a single line containing a single asterisk.
-w *n*	Specify the number of input bytes to be displayed and interpreted on each line that is output. If **-w** is not given, then it is assumed that 16 bytes are read for each display line. If *n* is not given, it is assumed to be 32.
-x	Display bytes as hexadecimal.

An upper-case format character specifies that the long or double precision form of the object is to be used.

The *offset* argument specifies the byte offset into the file where dumping is to begin. By default, the *offset* argument is interpreted in octal. A decimal radix is assumed if a period is append to the *offset* argument. A hexadecimal radix is assumed if *offset* begins with `0x` or `x`. If `b` is appended, *offset* is interpreted as a count of 512- (1024-) byte blocks. In the output, the radix of the displayed address is the same that of *offset*. If the *file* argument is omitted, *offset* must be preceded by a +.

label is interpreted as a pseudo-address for the first byte displayed. It is shown in parenthesis following the *offset*. It is usually used to indicate the real memory address in core images. Syntax for *label* is the same that of *offset*.

NOTE

Filenames cannot start with the “+” character, and a hexadecimal offset cannot be used for a block count. Only one file can be dumped at a time. The type of highlighting used with the **-a** option depends on the type of output device being used. On monochrome monitors the appropriate characters are underlined, on color monitors they are a different color than the normal text color. Some printers may produce results that are difficult to read when the **-aP** or **-ap** options are used.

EXAMPLE

```
od -x core +01000
```

SEE ALSO

Utility Programs - `flook`