OPEN 7

Command Reference Manual (H to P)

DPS7000/XTA
NOVASCALE 7000

Operating System: Subsystems

REFERENCE
47 A2 83US 01

DPS7000/XTA NOVASCALE 7000

OPEN 7

Command Reference Manual (H to P)

Operating System: Subsystems

April 1997

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 83US 01

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

Preface

OPEN 7 is a GCOS 7 subsystem which provides a gateway between UNIX machines (or computers supporting the standard TCP/IP protocol) and the DPS 7000 GCOS 7 operating system.

Via OPEN 7, you can connect to the GCOS 7 Interactive Operator Facility (IOF) or Transaction Driven Subsystem (TDS). You can also transfer or distribute files between GCOS 7 and UNIX systems.

INTENDED READERS

This manual is intended for UNIX system administrators who wish to administer OPEN 7 on a DPS 7000.

SCOPE AND OBJECTIVES

This manual is part of a set of three manuals which describes the available OPEN 7 commands. Among these commands:

- some are GCOS 7 commands specific to OPEN 7,
- some are UNIX-like commands specific to OPEN 7 and are described in these manuals,
- some are standard UNIX commands; they are not described in these manuals, but you are referred to the appropriate documents.
- the most often used commands are listed in this preface. The other commands described in this manual are used only in special cases, generally by the Service Center.

If you administer OPEN 7 on the DPS 7000 and UNIX systems, you already have the standard documents.

If you administer OPEN 7 on the DPS 7000 and a non-UNIX system, such as MS-DOS, you may find it useful to obtain the standard documents. They are listed in this preface.

STRUCTURE OF THE DOCUMENTS

This manual is the second of a set of three:

Chapter 1 to 9 list the OPEN 7 commands in alphabetical order, from the letter "H" to the letter "P".

OPEN 7 MOST COMMONLY USED COMMANDS

Command Type	Command Names
User Management	defuser passwd grpck pwck
Directory Management	mkdir rmdir chgrp chown chmod
Device Management	mknod
File System Management	mkdf lsdf rmdf mount umount mountall umountall fsck backfsck df du fsstat mkfs mklost+found ulimit_no_prot
Process and System Management	who ps wall killall kill fuser shutdown uadmin sync gnice gdate
GCOS 7 Interoperability	cndsa formc setdsa cpgtou cputog mkgfile lsgfile rmgfile lsufas mkufas
Initializing TCP/IP 7	hostname ifconfig ipx25 tcpadmin route
TCP/IP 7 Maintenance	netstat arp ruptime rwho lanspy
NFS Commands	exportfs
NFS Maintenance	showmount nfsstat rcpinfo
TCP/IP and NFS Daemons	inetd rlogind telnetd rwhod nfsd biod portmap
Managing Priorities and Dates	gnice gdate
Miscellaneous	autoreboot sysdef cpio

OPEN 7 SET OF DOCUMENTS

OPEN 7 User's Guide 47 A2 80US

OPEN 7 Administrator's Guide.....47 A2 81US
OPEN 7 Administrator's Reference Manual (Vol 1).....47 A2 82US
OPEN 7 Administrator's Reference Manual (Vol 2).....47 A2 83US
OPEN 7 Administrator's Reference Manual (Vol 3).....47 A2 84US

The User's Guide gives an overview of the OPEN 7 set of products, then describes the more common file-managing tasks that use OPEN 7 functions.

The Administrator's Guide describes how to install, configure and manage the OPEN 7 products.

The present Administrator's Reference Manuals list the commands available under OPEN 7 and the GCOS 7 commands used to manage OPEN 7.

ASSOCIATED STANDARD UNIX DOCUMENTS

The following UNIX System V documents are published by Prentice Hall, Inc.

UNIX System V User's Guide
UNIX System V User's Reference Manual
UNIX System V Programmer's Guide
UNIX System V Programmer's Reference Manual

Contents of the UNIX System V User's Guide

Part 1: UNIX System Overview

1. What is the UNIX System ?
2. Basics for UNIX System Users

Part 2: UNIX System Tutorials

2. Using the File System
4. Overview of the Tutorials
5. Line Editor Tutorial (ed)
6. Screen Editor Tutorial (vi)
7. Shell Tutorial
8. Communication Tutorial

Appendix A: Summary of the File System
Appendix B: Summary of UNIX System
Appendix C: Quick Reference to ed
Appendix D: Quick Reference to vi
Appendix E: Summary of Shell Commands
Appendix F: Setting Up the Terminal

Contents of the UNIX System V User's Reference Manual

Section (1,1M): Commands and Utilities

Contents of the UNIX System V Programmer's Guide

Contents of the UNIX System V Programmer's Reference Manual

Other Standard Documents

The C Programming Language by B. W. Kernighan.
Programming in C: A Tutorial by B. W. Kernighan.
C Reference Manual by D. M. Ritchie & B. W. Kernighan.

X/OPEN Documents

As OPEN 7 is fully compatible with X/OPEN, you may also consult the:

X/OPEN Portability Guide (5 Volumes).....13 A2 37TG

ASSOCIATED DPX/20 DOCUMENTS

For a description of TCP/IP, consult the following:

Bull DPX/20 System User's Guide for Operating System and Devices.....86 A2 51AP
Bull DPX/20 System Management Guide for Operating System and Devices86 A2 53AP
Bull DPX/20 System User's Guide for Communications and Networks 86 A2 52AP
Bull DPX/20 System Management Guide for Communications and Networks86 A2 54AP

Bull DPX/20 Commands Reference Manual (Vol 1) 86 A2 73AP
.
.
.
Bull DPX/20 Commands Reference Manual (Vol 6) 86 A2 78AP

ASSOCIATED AFFINITY DOCUMENTS

Supplement to the Installation and Setup Guide for Affinity/OPEN 7.....40 A2 13WA
Affinity V2 for Open7 - Front-End Installation and Configuration 40 A2 14WA

ASSOCIATED GCOS 7 DOCUMENTS

The following GCOS 7 manuals may be of particular interest:

GCOS 7 Information Directory..... 47 A2 11UG

Networks: Overview..... 47 A2 92UC
Networks: Generation 47 A2 93UC
Networks: User's Guide 47 A2 94UC

IOF Terminal User's Reference Manual (V7)
Introduction..... 47 A2 38UJ
GCL Commands 47 A2 39UJ
Directives and General Processor Commands 47 A2 40UJ

GCL Programmer's Manual.(V7) 47 A2 36UJ
IOF Programmer's Manual (V7)..... 47 A2 37UJ
JCL Reference Manual..... 47 A2 11UJ

UFAS-EXTENDED User's Guide.....47 A2 04UF

Library Maintenance Reference Manual 47 A2 01UP
Library Maintenance User's Guide.....47 A2 02UP

System Installation, Configuration, and Updating Guide.....47 A2 18US

DNS V4 NGL Reference Manual 39 A2 23DN
DNS V4 System Generation Manual 39 A2 22DN
CNS7 A2 NGL Reference Manual 39 A2 32DN

*SQL*NET with ORACLE-V6 User's Guide..... 47 A2 03UR*
*SQL*NET with ORACLE 7 User's Guide 47 A2 13UR*

NOTATION CONVENTIONS

The notation conventions used are those of UNIX. The object names are followed by a number (1 through 8), this number itself possibly followed by a letter. They refer to chapters and portions of chapters in the standard UNIX manuals (for example, *telnet* (1C)).

Table of Contents

1. Commands Beginning with the Letter "H"

1. head(1)

NAME

head - give first few lines

SYNOPSIS

```
head [ -count ] [ file ...  ]
```

DESCRIPTION

This filter gives the first count lines of each of the specified files, or of the standard input. If count is omitted it defaults to 10.

SEE ALSO

tail(1).

1. hostname(1)

NAME

hostname - sets or displays the name of the current host system

SYNOPSIS

```
hostname [name-of-host]
```

DESCRIPTION

The *hostname* command prints the name of the current host, by which the system is known to the network. The super-user can set the hostname by giving an argument; this is usually done in the startup script */etc/bcheckrc*.

EXAMPLES

```
hostname `uname -n`
hostname
```

APPLICATION NOTES

As most of the commands dealing with the network come from BSD UNIX, the BSD *hostname* command has been given preference over the ATT command *uname -n*. But, to be compatible with both versions of UNIX, the startup script */etc/bcheckrc* sets both values to be the same.

FILES

```
/etc/bcheckrc
```

SEE ALSO

uname(1), gethostname(2)

1. hosts(4)

NAME

hosts - list of hosts on network

SYNOPSIS

```
/etc/hosts
```

DESCRIPTION

The */etc/hosts* file is a list of hosts that share the network, including the local host. It is referred to by programs that need to translate between host names and DARPA Internet addresses when the name server (see *named(1M)*) is not being used.

Each line in the file describes a single host on the network and consists of three fields separated by any number of blanks or tabs:

```
address name aliases ...
```

where:

address	is the DARPA Internet address. Unless another type of address is required by some host on the network, address should be a Class A address, which takes the form *net.node*, where *net* is the network number from */etc/networks* (see *networks(4)*), which must be betwen 0 and 127 ; and *node* is a value which must be unique for each host and be between 0 and 16777215.
name	is the official name of the host. If the host is a computer system running UNIX, it must claim this host name by executing *hostname(1M)* when it is initializing itself.
aliases ...	is a list of alternate names for the host. Aliases can be used in network commands in place of the official name.

It is suggested that you specify the *hostname* and the *node* name (see *hostname(1)* and *uname(1)*) as aliases of one another for each machine listed in the */etc/hosts* file.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Internet addresses can actually take one of four forms:

A	A is a simple 32-bit integer.
A.B	A is an eight-bit quantity occupying the high-order byte and B is a 24-bit quantity occupying the remaining bytes. This form is suitable for a Class A address of the form *net.node.*
A.B.C	A is an eight-bit quantity occupying the high-order byte. B is an eight-bit quantity occupying the next byte C is a 16-bit quantiy occupying the remaining bytes. This form is suitable for a Class B address of the form *128.net.node*.
A.B.C.D	The four parts each occupy a byte in the address.

EXAMPLE

```
#       Engineering network

1.12    src.MySite.COM     src     net3    # Network Source Machine
1.10    test.MySite.COM    test    net2    # Network Test Machine
1.16    mifa.MySite.COM    mifa            # Software Development
1.17    mifb.MySite.COM    mifb            # Hardware Development
```

FILES

/etc/hosts

SEE ALSO

uname(1), hostname(1), networks(4), inet(7).

1. hosts.equiv(4)

NAME

hosts.equiv - list of INET trusted hosts

SYNOPSIS

```
/etc/hosts.equiv
```

DESCRIPTION

The */etc/hosts.equiv* file contains a list of trusted hosts. When an *rlogin(1)* or *rshell(1)* request from such a host is made, and the initiator of the request is in */etc/passwd*, then no further validity checking is done. That is, *rlogin* does not prompt for a password, and *rshell* completes successfully. So a remote user is "equivalenced" to a local user with the same user ID when the remote user is in *hosts.equiv*.

The format of *hosts.equiv* is a list of names as in this example (the host names must be the standard names as described in *rshell(1)*):

```
host1
host2
```

Programs scan *hosts.equiv* linearly. If the remote host is equivalenced by an entry, it means that anyone logging in from that host is trusted and the program stops.

When user XXX executes *rlogin* or *rshell*, the *.rhosts* file from XXX 's home directory is conceptually concatenated onto the end of *hosts.equiv* for permission checking.

The *.rhosts* file has the same format as *hosts.equiv*.

It is also possible to have two entries (separated by a single space) on a line of these files. In this case, if the remote host is equivalenced by the first entry, then the user named by the second entry is allowed to log in as anyone, that is, specify any name to the -l flag (provided that name is in the */etc/passwd* file, of course).

Thus

```
eiger john
```

allows john to log in from eiger as anyone. The usual usage would be to put this entry in the *.rhosts* file in the home directory for *bill*. Then *john* may log in as *bill* when coming from *eiger*.

APPLICATION USAGE

The *hosts.equiv* file can be readable by anyone, but should be writeable only by **root**.

Leave no trailing blanks in the file.

FILES

/etc/hosts.equiv

SEE ALSO

rhosts(4), rlogin(1), rshell(1), rcp(1) rlogind(1M), rshd(1M).

2. Commands Beginning with the Letter "I"

2. icmp(7)

NAME

icmp - Internet Control Message Protocol

SYNOPSIS

OPEN 7 configuration (file config.c):

```
/dev/inet/icmp   major 29   minor 32
```

programmer's interface:

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);
```

DESCRIPTION

ICMP is the error and control message (or device) protocol used by IP and the Internet protocol family. It may be accessed through a "raw socket" for network monitoring and diagnostic functions.

The *proto* parameter to the socket call to create an ICMP socket is obtained from *getprotobyname* (See getprotoent(3)).

ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls; the *connect(2)* call may also be used to fix the destination for future packets (in which case the *read(2)* or *recv(2)* and *write(2)* or *send(2)* system calls may be used).

Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOSR]	when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

FILES

/dev/inet/icmp

SEE ALSO

send(2), recv(2), intro(7), inet(7), ip(7).

2. id(1M)

NAME

id - print user and group IDs and names

SYNOPSIS

```
id
```

DESCRIPTION

The *id* command outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1), getuid(2)

2. ifconfig(1M)

NAME

ifconfig - configure network interface parameters for a network using TCP/IP

SYNOPSIS

```
/etc/ifconfig interface address_family [address [dest_address]]
[parameters]

/etc/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 pararneters. The *interface* parameter is a string of the form "name unit", e.g. "en0".

Since an interface may receive transmissions in differing protocols each of which may require separate naming schemes, it is necessary to specify the *address_family,* which may change the interpretation of the remaining parameters. Currently only the Internet address family is supported: thus, the only valid value for *address_family is* inet.

For the DARPA_Internet family, the address is either a host name present in the host name data base, *hosts(4),* or a DARPA Internet address expressed in the Internet standard "dot notation".

The following parameters may be set with *ifconfig:*

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.

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.

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 (see arp(4); 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.

-trailers
Disable the use of a "trailer" link level encapsulation (default).

Running without trailers is strongly encouraged, if possible, because the use of trailers slows performance, especially when receiving data with trailers.

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. This option is not applicable in the STREAMS environment.

Use of arp for an interface is specified in */etc/strcf*. The arp driver will be opened when the STREAMS stack is built.

-arp
Disable the use of the Address Resolution Protocol.

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

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

-debug
Disable driver dependent debugging code.

netmask *mask*
(Inet only) 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(4)*. 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.

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

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.

onepacket
Enable the *one-packet* mode of operation (used for interfaces that cannot handle back-to-back packets). The keyword onepacket must be followed by two numeric parameters, giving the small packet size and threshold, respectively. If small packet detection is not desired, these values should be zero. See tcp(7) for an explanation of one-packet mode.

-onepacket
Disable one-packet mode.

pointopoint
(Inet only) Set the interface to be a point-to-point link.

public
(Inet only) Disable sending user datagrams in broadcast mode (like the one sent by *rwhod(1M)* or *routed(1M))* for cost reasons on public networks.

x121address
(Inet only) Specify the X121 format address to use to link the interface unit to a local physical unit (X25 subscription for example).

xcug
Manages the Closed User Group (CUG) list to which the interface belongs.

CUG list management (add, remove and display) is only allowed on the super-user login. Use of this facility complies partially with the description made in the ISO 8208:1990 norm, compliance depends upon which type of network the interface is configured for:

- **private network:** if this facility is not provided by the under layer X25-3, a restricted implementation is provided by the X25/IP driver.

  It consists in using the CUG Selection facility in outgoing X25-3 Call Request Packet and checking the CUG identifier in incoming X25-3 Call Request Packet acording to those which the interface belongs to. This implementation is restricted in the sense that it does not provide security as described in ISO 8208:1990. Indeed anyone can configure a interface with any CUG identifier. So an interface on a given host which belongs to a given CUG may be reached by an unauthorized remote host, if this remote host tries to configure its source interface with the 9999 possible CUG identifiers (in extended format). Security leads on the assumption that the 9999 random retries (configure and call) are prohibitive enough.

- **public network:** this facility is generally provided by the public data network, in which case the only thing to do is to configure the interface with the CUG identifiers mentioned in the subscription sheet.

When an interface is configured with no CUG the common group is used (no use of the CUG Selection facility in X25-3 Call Request Packet). On the other hand, if the interface is configured with at least one CUG and belongs to the common CUG, it must be explicitly mentioned.

When an interface belongs to more than one CUG, a retry-mechanism is provided in the virtual circuit establishment phase: the first successful CUG is used. In order to optimize this establishment phase a given CUG can be first-tried when calling a given remote subscriber. This can be specified by way of the "route" administrative command.

When an interface is configured with only one CUG, the CUG identifier is used in the Call Request Packet whether the interface is configured for a public data network or a private one.

xcug [CUG_id...]
Adds one or several CUG identifiers.

Example to use the ix1 interface within the Closed User Group 87 and 29:

```
ifconfig ix1 xcug 87 29
```

-xcug [CUG_id...]
Removes one or several CUG idenfifiers.

Example to specify that ix 1 interface no longer belongs to the CUG 29:

```
ifconfig ix1 -xcug 29
```

xcug flush
Removes all CUG from an interface.

xcug bas|ext
The bas and ext options specify the format used in Call Request Packet for CUG indentifier:

bas	the basic format is used, the CUG identifier range is 0-99 coded on one byte in BCD (CUG Selection Basic Format facility),
ext	the extended format is used, the CUG identifier range is 0-9999 coded on two bytes in BCD (CUG Selection Extended Format facility).

The default format used for an interface is the extended one.

xmtu
Set the maximum transmission unit size for a X25/IP interface

xrvcin
Manages the list of subscribers (known by their X121 addresses) with which the Reverse Charging Facility (RVC) is accepted. Obviously this facility has no meaning when the interface is not configured with a public data network.

xrvcin [XI21-addr ...]
Adds one or several subscribers as accepted Reverse Charging Callers.

Example to specify that incoming calls from subscribers 100 and 400, with the reverse charging facility are accepted.

```
ifconfig ix1 -xrvcin 100 400
```

-xrvcin [X121_addr ...]

Removes one or several subscribers as accepted Reverse Charging Callers.

Example to specify that incoming reverse charged calls from subscriber 100 are no longer accepted.

```
ifconfig ix1 -xrvcin 100
```

xrvcin flush
Removes all Reverse Charging Caller subscribers from an interface.

xrvcout
Manages the list of subscribers (known by their X121 addresses) with which the Reverse Charging facility (RVC) is used. Obviously this facility has no meaning when the interface is not configured with a public data network.

xrvcout [X121-addr]
Adds one or several subscribers to the list with which the Reverse Charging facility is used.

Example to specify that the Reverse Charging facility has to be used when calling subscribers 300 and 700.

```
ifconfig ix1 xrvcout 300 700
```

-xrvcout [X121_addr]
Removes one or several subscribers from the list with which the Reverse Charging facility is used.

Example to specify that the Reverse Charging facility is no longer used when calling subscriber 300.

```
ifconfig ix1 -xrvcout 300
```

xrvcout flush
Removes all subscribers from the list with which the Reverse Charging facility is used.

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.

FILES

/etc/ls_attach calls ifconfig to start serial lines.

SEE ALSO

arp(1M), tcp(1M), netstat(1), hosts(4), networks(4), strcf(4), arp(7), netintro(7), tcp(7)

2. indent(1)

NAME

indent - indent and format a C program source file

SYNOPSIS

```
indent  input-file  [output-file] [-bap | -nbap] [-bacc | -nbacc]
        [-bad | -nbad] [-bbb | -nbbb] [-bc | -nbc] [-bl] [-br]
        [-bs | -nbs] [ -cn ] [-cdn ] [-cdb | -ncdb]  [-ce | -nce]
        [-cin] [-clin] [-dn] [-din]  [-eei | -neei] [-fc1 | -nfc1]
        [-in] [-ip | -nip] [-ln] [-lcn] [-lp | -nlp]
        [-pcs | -npcs] [-npro] [-psl | -npsl] [-sc | -nsc]
        [-sob | -nsob] [-st] [-troff] [-v | -nv]
```

DESCRIPTION

indent is a C program formatter. It reformats the C program in the input-file according to the switches. The switches which can be specified are described below. They may appear before or after the file names.

NOTE: if you only specify an input-file, the formatting is done "in-place", that is, the formatted file is written back into *input-file* and a backup copy of input-file is written in the current directory. If *input-file* is named */blah/blah/file*, the backup file is named *file.BAK*.

If *output-file* is specified, *indent* checks to make sure it is different from *input-file*.

OPTIONS

The options listed below control the formatting style imposed by *indent*.

-bap, -nbap	If -bap is specified, a blank line is forced after every procedure body. Default: -nbap.
-bacc, -nbacc	If -bacc is specified, a blank line is forced around every conditional compilation block, that is, in front of every *#ifdef* and after every *#endif*. Other blank lines surrounding these will be swallowed. Default: -nbacc.
-bad, -nbad	If -bad is specified, a blank line is forced after every block of declarations. Default: -nbad.
-bbb, -nbbb	If -bbb is specified, a blank line is forced before every block comment. Default: -nbbb.

-bc, -nbc

If -bc is specified, then a NEWLINE is forced after each comma in a declaration. -nbc turns off this option. The default is -bc.

-br, -bl

Specifying -bl lines up compound statements like (1). Specifying -br (the default) makes them look like (2).

```
if (...)            }  (1)
{                   }
code                }
}                   }


if (...) {          }  (2)
code                }
}                   }
```

-bs, -nbs

Enable (disable) the forcing of a blank after *sizeof*. Some people believe that *sizeof* should appear as though it were a procedure call (-nbs, the default) and some people believe that since *sizeof* is an operator, it should always be treated that way and should always have a blank after it.

-cn

The column in which comments on code start. The default is 33.

-cdn

The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

-cdb, -ncdb

Enable (disable) the placement of comment delimiters on blank lines. With this option enabled, comments look like (1) rather than like (2). This only affects block comments, not comments to the right of code. The default is -cdb.

```
/*                              }  (1)
* this is a comment             }
*/                              }


/* this is a comment */            (2)
```

-ce, -nce

Enables (disables) forcing *else*'s to cuddle up to the immediately preceding "}". The default is -ce.

-cin

Sets the continuation indent to be n. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless -lp is in effect. -ci defaults to the same value as -i.

-clin

Cause case labels to be indented n tab stops to the right of the containing switch statement. -cli0.5 causes case labels to be indented half a tab stop. The default is -cli0.

-dn	Control the placement of comments which are not to the right of code. The default -d1 means that such comments are placed one indentation level to the left of code. Specifying -d0 lines up these comments with the code. See the section on comment indentation below.
-din	Specify the indentation, in character positions, from a declaration keyword to the following identifier. The default is -di16.
-eei, -neei	If -eei is specified, and extra expression indent is applied on continuation lines of the expression part of *if()* and *while()*. These continuation lines will be indented one extra level - twice instead of just once. This is to avoid the confusion between the continued expression and the statement that follows the *if()* or *while()*. Default: -neei.
-fc1, -nfc1	Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading "/" is in column 1 have been carefully hand formatted by the programmer. In such cases, -nfc1 should be used. The default is -fc1.
-in	The number of spaces for one indentation level. The default is 4.
-ip, -nip	Enables (disables) the indentation of parameter declarations from the left margin. The default is -ip.
-ln	Maximum length of an output line with a trailing comment. The default is 78.
-lcn	Sets the line length for block comments to n. It defaults to being the same as the usual line length as specified with -l.
-lp, -nlp	Lines up code surrounded by parenthesis in continuation lines. If 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. Example (1) shows how a piece of continued code looks with -nlp in effect. With -lp in effect (the default) the code looks somewhat clearer (2). Inserting a couple more NEWLINE characters we get (3).

```
p1 = first_procedure(second_procedure(p2, p3),          }  (1)
                     third_procedure(p4, p5));          }


p1 = first_procedure(second_procedure(p2, p3),          }  (2)
                     third_procedure(p4, p5));          }


p1 = first_procedure(second_procedure(p2,               }  (3)
                                      p3),              }
                     third_procedure(p4,                }
                                     p5));              }
```

-npro	Ignore the profile files, *./.indent.pro* and *~/.indent.pro*.
-pcs, -npcs	If true (-pcs) all procedure calls will have a space inserted between the name and the "(". The default is -npcs.
-psl , -npsl	If true (-psl) the names of procedures being defined are placed in column 1 - their types, if any, will be left on the previous lines. The default is -psl.
-sc, -nsc	Enables (disables) the placement of asterisks (*) at the left edge of all comments.
-sob, -nsob	If -sob is specified, *indent* will swallow optional blank lines. You can use this to get rid of blank lines after declarations. Default: -nsob.
-st	*indent* takes its input from the standard input, and put its output to the standard output.
-T typename	Add typename to the list of type keywords. Names accumulate: -T can be specified more than once. You need to specify all the typenames that appear in your program that are defined by *typedefs* - nothing will be harmed if you miss a few, but the program won't be formatted as nicely as it should. This sounds like a painful thing to have to do, but it is really a symptom of a problem in C: *typedef* causes a syntactic change in the language and indent cannot find all *typedefs*.
-troff	Causes indent to format the program for processing by *troff*. It will produce a fancy listing in much the same spirit as *vgrind*. If the output file is not specified, the default is standard output, rather than formatting in place. The usual way to get a troffed listing is with the command: `indent -troff program.c | troff -mindent`
-v, -nv	-v turns on "verbose" mode, -nv turns it off. When in verbose mode, *indent* reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is -nv.

FURTHER DESCRIPTION

You may set up your own "profile" of defaults to indent by creating a file called *.indent.pro* in either your login directory or the current directory and including whatever switches you like. An *.indent.pro* in the current directory takes precedence over the one in your login directory. If *indent* is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, always override profile switches. The switches should be separated by SPACE, TAB, or NEWLINE characters.

Comments

Boxed	*indent* assumes that any comment with a dash or star immediately after the start of comment (that is, /* or /**) is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.
Straight text	All other comments are treated as straight text. *indent* fits as many words (separated by SPACE, TAB, or NEWLINE characters) on a line as possible. Blank lines break paragraphs.
Comment indentation	If a comment is on a line with code it is started in the "comment column", which is set by the -cn command line parameter. Otherwise, the comment is started at n indentation levels less than where code is currently being placed, where n is specified by the -dn command line parameter. 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.

Preprocessor lines

In general, *indent* leaves preprocessor lines alone. The only reformatting that it will do is to straighten up trailing comments. It leaves imbedded comments alone. Conditional compilation (*#ifdef...#endif*) is recognized and *indent* attempts to correctly compensate for the syntactic peculiarities introduced.

C syntax

indent understands a substantial amount about the syntax of C, but it has a "forgiving" parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like the following one is handled properly:

```
#define forever for(;;)
```

FILES

./.indent.pro	profile file
~/.indent.pro	profile file
/usr/share/lib/tmac/tmac.indent	troff macro package for 'indent -troff' output.

SEE ALSO

ls(1), troff(1)

BUGS

indent has even more switches than *ls(1V)*.

A common mistake that often causes grief is typing the following command to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.

```
indent *.c
```

The -bs option splits an excessively fine hair.

2. inet(7)

NAME

inet - Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The Internet protocol family is a set of protocols using the Internet Protocol (IP) network layer and the Internet address format.

The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

ADDRESSING

Internet addresses are four-byte quantities, stored in network standard format. The include file *< sys/in.h >* defines this address as a discriminated union.

Sockets bound to the Internet protocol family use the following addressing structure:

```
struct sockaddr_in {
   short     sin_family;
   u_short   sin_port;
   struct    in_addr sin_addr;
   char      sin_zero[8];
};
```

When using sockets, the *sin_family* field is specified in host order, and the *sin_port* and *sin_addr* fields are specified in network order.

Sockets may be created with the local address INADDR_ANY to affect wildcard matching on incoming messages. The address in a *connect(2)* or *sendto* (see *send(2)*) call may be given as INADDR_ANY to mean "`this host."

The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

When using the Transport Layer Interface (TLI), transport providers such as *tcp(7)* support addresses whose length varies from eight to sixteen bytes. The eight byte form is the same as a *sockaddr_in* without the *sin_zero* field. The sixteen byte form is identical to *sockaddr_in*.

Additionally, when using TLI, the *sin_family* field is accepted in either host or network order. For communicating with other implementations via RFS, the preferred form is eight bytes with *sin_family* in network order.

PROTOCOLS

The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP).

TCP is used to support the SOCK_STREAM abstraction; UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part.

Sites with a cluster of local networks and a connection to the DARPA Internet may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts.

Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry.

Subnet addressing is enabled and examined by the following *ioctl(2)* commands on a datagram socket in the Internet "communications domain"; they have the same form as the SIOCIFADDR command (see *intro(7)*).

SIOCSIFNETMASK	Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.
SIOCGIFNETMASK	Get interface network mask.

SEE ALSO

ioctl(2), socket(2), intro(7), icmp(7), ip(7), tcp(7), udp(7).

CAVEAT

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

2. inetadm

A command used by the network administrator to manage the TCP/IP network. It displays menus to add or remove hardware interfaces (that is to update the */etc/interfaces* file) and to add or remove hosts, networks and routes, that is to update the */etc/hosts* and */etc/networks* files, and execute the */etc/route add* or */etc/route delete* commands.

Its use is described in chapter 4, "*TCP/IP Administration*", of the *OPEN 7 Administrator's Guide*.

2. inetd(1M)

NAME

inetd - internet "super-server"

SYNOPSIS

```
/etc/inetd [-d] [ configuration file ]
```

DESCRIPTION

inetd
listens on multiple ports for incoming connection requests. When it receives a request, it spawns the appropriate server. The use of a "super-server" allows other servers to be spawned only when needed and to terminate when they have satisfied a particular request.

The mechanism is as follows: *inetd* is started by the super-user (usually during init 2). To obtain information about the servers it needs to spawn, *inetd* reads its configuration file (by default, this is */etc/inetd.conf*(4)) and issues a call to *getservbyname* (see *getservent*(3)). (Note that **/etc/services** and **/etc/protocols** must be properly configured.) *inetd* then creates a socket for each server and binds each socket to the port for that server. It does a *listen*(2) on all connection-based sockets (that is, stream rather than datagram), and waits, using *select*(2), for a connection or datagram.

- When a connection request is received on a listening (stream) socket, *inetd* does an *accept*(2), thereby creating a new socket. (*inetd* continues to listen on the original socket for new requests). *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments specified in *inetd's* configuration file. The invoked server has I/O to *stdin*, *stdout*, and *stderr* done to the new socket; this connects the server to the client process. (Some "built-in" internal services are performed via function calls rather than child processes.)

- When there is data waiting on a datagram socket, *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments; unlike a connection-based server, a datagram server has I/O to *stdin*, *stdout*, and *stderr* done to the original socket. If the datagram socket is marked as "wait" (this corresponds to an entry in *inetd's* configuration file), the invoked server must process the message before *inetd* considers the socket available for new connections. If the datagram socket is marked as "nowait," *inetd* continues to process incoming messages on that port. *tftpd* is an exceptional case: although its entry in *inetd's* configuration file must be "wait" (this is to avoid contention for the port), *inetd* is able to continue processing new messages on the port.

The following servers may be started by *inetd* : *fingerd, ftpd, rexecd*, *rlogind*, *rshd, talkd*, *telnetd,* and *tftpd*. *inet* must also start several internal services: these are described in *inetd.conf*(4).

Do not arrange for *inetd* to start *named*, *routed*, *rwhod*, *sendmail*, *slipd*, *listen* (RFS listening server), or any NFS server.

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.

The **-d** option turns on socket-level debugging and prints debugging information to *stdout*.

FILES

```
/etc/inetd.conf
/etc/protocols
/etc/services
```

SEE ALSO

fingerd(1M), ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), talkd(1M), telnetd(1M), tftpd(1M), inetd.conf(4), protocols(4), services(4).

2. inetd.conf(4)

NAME

inetd.conf - configuration file for inetd (internet "super-server")

DESCRIPTION

inetd.conf is the configuration file for the inetd(1M) System V STREAMS TCP/IP internetworking "super-server".

The file consists of a series of single-line entries, each entry corresponding to a service to be invoked by inetd. These services are connection-based, datagram, or ``internal''.

Internal services are those supported by the inetd program: 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, January 1, 1900). All of these services are tcp based. (For details of these services, consult the appropriate RFC from the DDN Network Information Center.)

Each service, including internal services, must have a valid entry in */etc/services(4)*. In the case of an internal service, its name must correspond to the official name of the service: that is, the first entry in */etc/services*.

Each entry has a series of space- or tab-separated fields. No field, except for the last one, may be omitted. The fields are as follows:

service name	Name of a valid service in */etc/services*, as described above.
socket type	One of "stream", "dgram", or "raw", depending on whether the socket type is stream, datagram, or raw (see *socket(2)*).
protocol	Name of a valid protocol (for example, "tcp") specified in */etc/protocols(4)*.
wait/nowait	Specifies whether the socket can be made available for new connections while there is still data waiting on the socket. The value is always "nowait" unless it is a datagram socket. If it is a datagram socket, the value is usually "wait", although "nowait" is possible in some cases. (Note that tftpd is an exception in that it must have "wait" specified, and yet the socket can continue to process messages on the port.)
user	Name of the user as whom the server should run. This allows servers to be run with less permission than root.

server program — Except in the case of internal services, full pathname of the server program to be invoked by inetd when a request is waiting on a socket. For an internal service, the value is "internal".

server program arguments — Arguments to the server program, starting with argv[0], which is the name of the program. For an internal service, the value is "internal".

Comments are denoted by a "#" at the beginning of a line.

The distribution *inetd.conf* file contains prototype entries; refer to these entries when editing the file.

EXAMPLE

```
                    .
                    .
                    .
ftp       stream   tcp   nowait   root   /etc/ftpd      ftpd
telnet    stream   tcp   nowait   root   /etc/telnetd   telnetd
login     stream   tcp   nowait   root   /etc/logind    logind
exec      stream   tcp   nowait   root   /etc/execd     execd
finger    stream   tcp   nowait   root   /etc/fingerd   fingerd
echo      stream   tcp   nowait   root   internal
discard   stream   tcp   nowait   root   internal
chargen   stream   tcp   nowait   root   internal
daytime   stream   tcp   nowait   root   internal
time      stream   tcp   nowait   root   internal
echo      dgram    udp   wait     root   internal
discard   dgram    udp   wait     root   internal
chargen   dgram    udp   wait     root   internal
daytime   dgram    udp   wait     root   internal
time      dgram    udp   wait     root   internal
                    .
                    .
                    .
```

SEE ALSO

fingerd(1M), ftpd(1M), inetd(1M), rexecd(1M), rlogind(1M), rshd(1M), telnetd(1M), tftpd(1M), protocols(4), services(4).

2. infocmp(1M)

NAME

infocmp - compares or prints out terminfo descriptions

SYNOPSIS

```
infocmp [-d] [-c] [-n] [-I] [-L] [-C] [-r] [-u] [-s d|i|l|c] [-v]
[-V] [-l] [-w width] [-A directory] [-B directory] [termname ...]
```

DESCRIPTION

infocmp can be used to compare a binary *terminfo(4)* entry with other terminfo entries, rewrite a *terminfo(4)* description to take advantage of the **use**=terminfo field, or print out a *terminfo(4)* description from the binary file *term(4)* in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified, and zero or one *termnames* are specified, the **-I** option will be assumed. If more than one *termname* is specified, the **-d** option will be assumed.

Comparison Options [-d] [-c] [-n]

infocmp compares the *terminfo(4)* description of the first terminal *termname* with each of the descriptions given by the entries for the other terminals' *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of capability: **F** for boolean variables, **-1** for integer variables, and **NULL** for string variables.

-d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal, and one wants to see what is different between the two.

-c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **-u** option is worth using.

-n produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable TERM will be used for both the *termnames*. This option can be used as a quick check to see if anything was left out of the description.

Source Listing Options [-I] [-L] [-C] [-r]

The **-I**, **-L**, **-C** options will produce a source listing for each terminal named.

-I uses the *terminfo(4)* names
-L use the long C variable name listed in **<term.h>**
-C use the *termcap* names
-r when using -C, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the -C option may be used directly as a *termcap* entry, but not all the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it does not will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo(4)*, but which are derivable from other *terminfo(4)* variables, will be output. Not all *terminfo(4)* capabilities will be translated; only those variables which were part of of *termcap* will normally be output. Specifying the -r option will take off this restrictiopn, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo(4)* string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo(4)* format will not necessarily reproduce the original *terminfo(4)* source.

Some common *terminfo* parameter sequences, their *termcap* equivalent, and some terminal types which commonly have such sequences, are:

```
Terminfo                    Termcap   Repres. Terminals

%p1%c                       %.        adm
%p1%d                       %d        hp, ANSI standard, vt100
%p1%'x'%+%c                 %+x       concept
%i                          %i        ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;    %>xy      concept
%p2 is printed before %p1   %r        hp
```

Use= Option [-u]

-u produce a *terminfo(4)* source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals' *termnames*. It does this by analysing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic *terminfo* entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termnames* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termnames* entries, or if the first of the other *termnames* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termnames* entries is significant. Since the terminfo compiler *tic(1M)* does a left-to-right scan of the capabilities, specifying two use= entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termnames* entries as they are found.

Alternatively, specifying a capability after a use= entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compile files, but will slow down the compilation time, is specifying extra use= fields that are superfluous. *infocmp* will flag any other termname use= fields that were not needed.

Other Options [-s d | i | l | c] [-v] [-V] [-1] [-w width]

-s sort the fields within each type according to the argument below:

d leave fields in the order that they are stored in the terminfo database.
i sort by terminfo name
l sort by the long C variable name.
c sort by the *termcap* name.

If no -s option is given, the fields printed out will be sorted alphabetically by the terminfo name within each type, except in the case of the -C or -L options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

-v print out tracing information on standard error as the program runs.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to be printed out one to a line. Otherwise, the fields will be printed several to a line, to a maximum width of 60 characters.

-w change the output to width characters.

Changing Databases [-A directory] [-B directory]

The location of the compiled *terminfo(4)* database is taken from the environment variable **TERMINFO**. If the variable is not defined, or the terminal is not found in that location, the system *terminfo(4)* database, usually in */usr/lib/terminfo*, will be used.

The options **-A** and **-B** may be used to override this location. The **-A** option will set TERMINFO for the first *termname* and the **-B** option will set TERMINFO for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases.

This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo(4)* database for a comparison to be made.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

DIAGNOSTICS

malloc is out of space!
There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use=order dependancy found:
A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=term' did not add anything to the description
A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison done
The **-u**, **-d** and **-c** options require at least two terminal names.

SEE ALSO

tic(1M), captoinfo(1M).

curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.

NOTES

The *termcap* database (from earlier releases of operating system) may not be supplied in future releases.

2. init(1M), telinit(1M)

NAME

init, telinit - process control initialization

SYNOPSIS

```
init [ 0123456SsQqabc ]

telinit [ 0123456SsQqabc ]
```

DESCRIPTION

init is a general process spawner. Its primary role is to create processes from information stored in the file */etc/inittab* (see inittab (4)).

At any given time, the system is in one of eight possible run levels.

A run level is a software configuration of the system under which only a selected group of processes exist.

The processes spawned by *init* for each of these run levels is defined in */etc/inittab*. *init* can be in one of eight run levels, 0-6 and S or s (run levels S and s are identical). The run level changes when a privileged user runs */etc/init*. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to init.

0	shut the machine down so it is safe to remove the power. Have the machine remove power if it can.
1	put the system in single-user mode. Unmount all file systems except root. All user processes are killed except those connected to the console.
2	put the system in multi-user mode. All multi-user environment terminal processes and daemons are spawned. This state is commonly referred to as the multi-user state.
3	reserved
4	is available to be defined as an alternative multiuser environment configuration. It is not necessary for system operation and is usually not used.
5	Stop the operating system and and reboot automatically OPEN 7.

6	reserved
a, b, c	process only those */etc/inittab* entries having the a, b or c run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.
Q, q	re-examine */etc/inittab*.
S, s	enter single-user mode. When this occurs, the terminal which executed this command becomes the system console. This is the only run level that does not require the existence of a properly formatted */etc/inittab* file. If this file does not exist, then by default the only legal run level that init can enter is the single-user mode. When the system enters S or s, all mounted file systems remain mounted and only processes spawned by *init* are killed.

When a operating system is booted, init is invoked and the following occurs.

First, *init* looks in */etc/inittab* for the *initdefault* entry (see *inittab(4)*).

- If there is one, *init* uses the run level specified in that entry as the initial run level to enter.
- If there is no *initdefault* entry in */etc/inittab*, *init* requests that the user enter a run level from the virtual system console.

If an S or s is entered, *init* goes go the single-user state. In the single-user state the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The command */bin/su* is invoked and a message is generated on the physical console saying where the virtual console has been relocated.

Use either *init* or *telinit*, to signal *init* to change the run level of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the single-user state if the */etc/inittab* file does not exist.

If a 0 through 6 is entered, *init* enters the corresponding run level. Note that the run levels 0, 1, and 5 are reserved states for shutting the system down; the run levels 2 and 4 are available as normal operating states.

If this is the first time since power up that *init* bas entered a run level other than single-user stage, *init* first scans */etc/inittab* for boot and bootwait entries (see inittab(4)). These entries are performed before any other processing of */etc/inittab* takes place, providing that the run level entered matches that of the entry. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. *init* then scans */etc/inittab* and executes all other entries that are to be processed for that run level.

In a multi-user environment, */etc/inittab* is set up so that init will create a *getty* process for each terminal that the administrator sets up to respawn.

To spawn each process in */etc/inittab*, *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by */etc/inittab*, *init* waits for one of its descendant process to die, a powerfail signal, or a signal from another *init* or *telinit* process to change the system's run level.

When one of these conditions occurs, *init* re-examines */etc/inittab*. New entries can be added to */etc/inittab* at any time; however, *init* still waits for one of the above three conditions to occur before re-examining /etc/inittab. To get around this, init Q or init q command wakes *init* to re-examine */etc/inittab* immediately.

When *init* comes up at boot time and whenever the system changes from the single-user state to another run state, *init* sets the *ioctl(2)* states of the virtual console to those modes saved in the file */etc/ioctl.syscon*. This file is written by *init* whenever the single-user state is entered.

When a run level change request is made, *init* sends the warning signal (SIGTERM) to all processes that are undefined in the target run level. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (SIGKILL).

The shell running on each terminal will terminate when the user types an end-of-file or hangs up. When *init* receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in */etc/utmp* and */etc/wtmp* if il exists (see *who(1)*). A history of the processes spawned is kept in */etc/wtmp*.

If *init* receives a powerfail signal (SIGPWR) it scans */etc/inittab* for special entries of the type powerfail and powerwait. These entries are invoked (if the run levels permit) before any any further processing takes place. In this way *init* can perform various cleanup and recording fonctions during the powerdown of the operating system. Note that in the single-user states, S and s, only powerfail and powerwait entries are executed.

telinit

telinit, which is linked to */etc/init*, is used to direct the actions of *init*. It takes a one-character argument and signals *init* to take the appropriate action.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/console
/dev/systty

NOTES

C2 Secure Environment
In a Secure environment, the *init* command has been moved to the */tcb/bin* directory.

SEE ALSO

getty(1M), shutdown(1M), termio(7), login(1), sh(1), stty(1), who(1), kill(2), gettydefs(4), inittab(4), utmp(4)

DIAGNOSTICS

If init finds that it is respawning an entry from */etc/inittab* more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console.

It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in */etc/inittab*.

When attempting to boot the system, failure of *init* to prompt for a new run level may be because the virtual system console is linked to a device other than the physical system console.

WARNINGS

init and *telinit* can be run only by someone who is super-user.

The S or s state must not be used indiscriminately in the */etc/inittab* file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the initdefault.

The change to */etc/gettydefs* described in the **WARNINGS** section of the *gettydefs(4)* manual page will permit terminals to pass 8 bits to the system as long as the system is in multi-user state (run level greater than 1). When the system changes to single-user state, the getty is killed and the terminal attributes are lost.

To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type:

```
stty -istrip cs8
```

2. initlp(1M)

NAME

initlp - initializes the LP spooling system

SYNOPSIS

```
/etc/initlp
```

DESCRIPTION

The *initlp* procedure must be included in the file */etc/rc2.d/S99initlp* in order to be called automatically by *rc2*(1M) at OPEN 7 boot time.

If no OPEN 7 printers have been installed using the *lpsetup* (1M) command, *initlp* initializes only printers of the SYSOUT class which allow OPEN 7 reports to be printed on GCOS 7 printers.

If OPEN 7 printers have been installed using *lpsetup* (1M), they are also started by *initlp*.

FILES

`/etc/config.lp` contains the description of printers to be started by *initlp*

SEE ALSO

lpsetup(1M), lpsched(1M), lpshut(1M)

2. inittab(4)

A file which stores information about processes. It is used by init(1M) to create these processes. See the *Programmer's Reference Manual*.

2. install(1M)

NAME

install - install commands

SYNOPSIS

```
install [-c dira] [-f dirb] [-i] [-n dirc] [-m mode] [-u user]
        [-g group] [-o] [-s] file [dirx ...]
```

DESCRIPTION

The *install* command is most commonly used in "makefiles" (see *make(1)*) to install a file (updated target file) in a specific place within a file system. Each file is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command.

The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx ...*) are given, *install* will search a set of defaults directories (/bin, /usr/bin, /etc, /lib and /usr/lib, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx ...*) are specified after file, those directories will be searched before the directories specified in the default list.

The meaning of the options are:

-c dira	installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. This option may be used alone or with the -s option.
-f dirb	Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to *755* and *bin*, respectively. If the file already exists, the mode and owner will be that of the existing file. May be used alone or with the -o or -s options.

-i	Ignores default directory list, searching only through the given directories (*dirx ...*). May be used alone or with any other options except -c and -f.
-n dirc	If file is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to *755* and *bin*, respectively. May be used alone or with any other options except -c and -f.
-m mode	The mode of the new file is set to *mode*. Only available to the superuser.
-u user	The owner of the new file is set to *user*. Only available to the superuser.
-g group	The group id of the new file is set to *group*. Only available to the superuser.
-o	If *file* is found, this option saves the "found" file by copying it to *OLDfile* in the directory in which it was found. This option is useful when installing a frequently used file such as *bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except -c.
-s	Suppresses printing of messages other than error messages. May be used alone or with any other options.
file	The file to copy into the appropriate directory (specified by *dirx ...*, or one of the default list).
dirx ...	One or more directories to be searched before the directories specified in the default list.

SEE ALSO

make(1P)

2. interfaces(4)

NAME

interfaces - TCP/IP 7 data base interfaces

SYNOPSIS

```
/etc/interfaces
```

DESCRIPTION

The *interfaces* file contains information regarding the interfaces on your machine, for use by the *tcpadmin*(1M) command. For each interface, a single line should be present with the following information:

- name of the interface
- hostname associated with this interface

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment, after which characters up to the end of the line are ignored by *tcpadmin*.

EXAMPLE

```
#
# interfaces data base
#

# loopback driver
lo0     localhost       # Do not change this.

# ethernet adapter
ea12    mydps7          # Your own configuration.
```

SEE ALSO

tcpadmin(1M), ifconfig(1M)

2. ip(7)

NAME

ip - Internet Protocol

SYNOPSIS

OPEN 7 configuration (file config.c):

```
/dev/inet/ip   major 29   minor 33
```

programmer's interface:

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);
```

DESCRIPTION

IP is the network layer protocol used by the Internet protocol family.

Options may be set at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). It may also be accessed through a "raw socket" or device when developing new protocols or special purpose applications. A single generic option is supported at the IP level, IP_OPTIONS, that may be used to provide IP options to be transmitted in the IP header of each outgoing packet.

Options are set with *setsockopt* and examined with *getsockopt* (see *getsockopt(2)*). The format of IP options to be sent is that specified by the IP protocol specification, with one exception: the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways.

The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use.

IP options may be used with any socket type in the Internet family.

Raw IP sockets are connectionless, and are normally used with the sendto and recvfrom calls; the *connect(2)* call may also be used to fix the destination for future packets (in which case the *read(2)* or *recv(2)* and *write(2)* or *send(2)* system calls may be used).

If *proto* is 0, the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If *proto* is non-zero, that protocol number will be used on outgoing packets and to filter incoming packets. *proto* must be specified in *sockcf(4)*.

Outgoing packets automatically have an IP header prepended to them (based on the destination address given and the protocol number the socket is created with). Incoming packets are received with IP header and options intact.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket has not been connected
[ENOSR]	when the system runs out of memory for an internal data structure
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

The following errors specific to IP may occur when setting or getting IP options:

[EINVAL]	An unknown socket option name was given.
[EINVAL]	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

SEE ALSO

getsockopt(2), send(2), recv(2), sockcf(4), intro(7), icmp(7), inet(7)

2. ipcrm(1)

NAME

ipcrm - remove a message queue, semaphore set, or shared memory ID

SYNOPSIS

```
ipcrm [ primitives ]
```

DESCRIPTION

ipcrm removes one or several messages, semaphores, or shared memory identifiers, as specified by the following primitives:

-q msqid	removes the message queue identifier *msqid* from the system and destroys the message queue and data structures associated with it.
-m shmid	removes the shared memory identifier *shmid* from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.
-s semid	removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structures associated with it.
-Q msgkey	removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structures associated with it.
-M shmkey	removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.
-S semkey	removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structures associated with it.

The identifiers and keys may be found by using *ipcs(1)*.

The details of removing identifiers are described in *msgctl(2)*, *shmctl(2)*, and *semctl(2)* in the sections detailing the IPC_RMID command.

SEE ALSO

ipcs(1), msgctl(2), msgget(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2)

2. ipcs(1)

NAME

ipcs - report interprocess communication facilities status

SYNOPSIS

```
ipcs [ primitives ]
```

DESCRIPTION

ipcs prints information about active interprocess communication facilities as specified by the primitives shown below. If no primitives are given, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

Command - Line Primitives

If any of the primitives -m, -q, or -s are specified, information about only indicated facilities is printed. If none of these are specified, information about all three is printed.

-q	Print information about active message queues.
-m	Print information about active shared memory segments.
-s	Print information about active semaphores.
-b	Print the currently allowed size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for the meaning of columns in a listing.
-c	Print creator's login name and group name. See below.
-o	Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)
-p	Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments). See below.

-t	Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* (see *msgop(2)*) on message queues, last *shmat* and last *shmdt* (see *shmop(2)*) on shared memory, last *semop(2)* on semaphores). See below.
-a	Use all display primitives. (This is a shorthand notation for -b, -c, -o, -p, and -t.)
-C corefile	Use the file *corefile* in place of */dev/mem*.
-N namelist	The argument will be taken as the name of an alternate file *namelist* (*/vmunix* is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the primitives that cause the corresponding heading to appear; all means that the heading always appears.

NOTE: these primitives only determine what information is provided for each facility; they do not determine which facilities will be listed.

T (all)	Type of the facility: q message queue m shared memory segment s semaphore
ID (all)	The identifier for the facility entry.
KEY (all)	The key used as an argument to *msgget(2)*, *semget(2)*, or *shmget(2)* to create the facility entry. Note that the key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.
MODE (all)	The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows: The first two characters are: R If a process is waiting on a msgrcv. S If a process is waiting on a msgsnd. D If the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it. C If the associated shared memory segment is to be cleared when the first attach is executed. - If the corresponding special flag is not set.

	The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows: r If read permission is granted. w If write permission is granted. a If alter permission is granted. - If the indicated permission is not granted.
OWNER (all)	The login name of the owner of the facility entry.
GROUP (all)	The group name of the group of the owner of the facility entry.
CREATOR(a,c)	The login name of the creator of the facility entry.
CGROUP (a,c)	The group name of the group of the creator of the facility entry.
CBYTES (a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM (a,o)	The number of messages currently outstanding on the associated message queue.
QBYTES (a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID (a,p)	The process ID of the last process to send a message to the associated queue.
LRPID (a,p)	The process ID of the last process to receive a message from the associated queue.
STIME (a,t)	The time the last message was sent to the associated queue.
RTIME (a,t)	The time the last message was received from the associated queue.
CTIME (a,t)	The time when the associated entry was created or changed.
NATTCH (a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ (a,b)	The size of the associated shared memory segment.
CPID (a,p)	The process ID of the creator of the shared memory entry.
LPID (a,p)	The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME (a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS (a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME (a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/vmunix	system namelist
/dev/mem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

ipcrm(1), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2)

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

2. ipx25(1M)

NAME

ipx25 - configure X25 interface for IP support

SYNOPSIS

```
/etc/ipx25 interface name address
/etc/ipx25 interface x25 x25_locaddr x25_remaddr [command]
/etc/ipx25 interface cv number_of_cv
/etc/ipx25 interface mtu max_transmission_unit
/etc/ipx25 interface mql max_send_queue_length
```

DESCRIPTION

ipx25 is used to configure network interfaces for IP support on an X25 network. The network interfaces used for IP/X25 are only logical point-to-point interfaces. The interface parameter is a string in the form "xai", where a is a letter in the range a through z, and i is a digit in the range 0 through 9, for example *xa0* or *xc2*.

The following parameters may be set with *ipx25*:

cv	sets the maximum number of virtual circuits to be opened for the specified interface. See Application Notes below.
name	sets the address of the host or network to be reached by any data output through this interface. The address is either a host name present in the host name database, *hosts* (4), or a network name present in the network name database, *networks* (4), or a DARPA Internet address expressed in the Internet standard dot notation.
x25	specifies a pair of X25 addresses to be used when opening a virtual circuit. *x25_locaddr* is a reference to an entry in the file */etc/confx25*(4) which describes a local output address. and *x25_remaddr* is the remote input address. *command* may be *add* or *delete*, to add or delete the pair of X25 addresses. The default is *add*.
mtu	sets the maximum number of bytes that can be sent by IP over the X25.4 interface. See Application Notes below.
mql	sets the maximum number of datagrams that can be queued on the interface outgoing flow. See Application Notes below.

When no parameters are supplied, *ipx25* displays the current X25 configuration for a network interface.

SEE ALSO

ifconfig(1M), routed(1M), route(1M), netstat(1), network(1M), xa(7)

APPLICATION NOTES

To configure an X25 interface, you must use at least two *ipx25* commands:

- to name the remote address,
- to associate a pair of X25 addresses to be used.

cv is set to 1 by default. A value greater than 1 may force upper layers, such as TCP, to spend time sorting incoming data protocol units.

mtu is present only for test purposes. It must be set to 575.

mql must be in the range 7 through 63. However, a value between 15 and 47 should be used for most applications. The choice must take into account that, if a datagram is not queued, it is dropped, and that datagrams must be acknowledged by TCP or another layer within a certain time. If the queue is too long and line throughput is poor, a datagram may be sent even though an upper layer has considered it to be lost.

3. Commands Beginning with the Letter "J"

3. join(1)

NAME

join - relational database operator

SYNOPSIS

```
join [-an] [-e string] [-j [1|2] m] [-o list] [-tc]
     filename1 filename2
```

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *filename1* and *filename2*. If *filename1* is "-", the standard input is used.

filename1 and *filename2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined - normally the first in each line.

There is one line in the output for each pair of lines in *filename1* and *filename2* that have identical join fields.

The output line normally consists of the common field, then the rest of the line from *filename1*, then the rest of the line from *filename2*.

The default input field separators are SPACE, TAB, and NEWLINE characters. If the default input field separators are used, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

OPTIONS

-an	The parameter n can be one of the values: 1 Produce a line for each unpairable line in *filename1*. 2 Produce a line for each unpairable line in *filename2*. 3 Produce a line for each unpairable line in both *filename1* and *filename2*. The normal output is also produced.
-e string	Replace empty output fields by *string*.
-j[1|2] m	The j may be immediately followed by n, which is either a 1 or a 2. If n is missing, the join is on the m'th field of both files. If n is present, the join is on the m'th field of file n, and the first field of the other. Note: *join* counts fields from 1 instead of 0 as *sort(1V)* does.
-o list	Each output line comprises the fields specified in *list*, each element of which has the form n.m, where n is a file number and m is a field number. The common field is not printed unless specifically requested. Note: *join* counts fields from 1 instead of 0 as *sort(1V)* does.
-tc	Use character c as a separator (tab character). Every appearance of c in a line is significant. The character c is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join -j1 4  -j2 3  -o 1.1 2.1 1.6  -t:  /etc/passwd  /etc/group
```

SEE ALSO

awk(1), comm(1), look(1), sort(1), uniq(1)

BUGS

With default field separation, the collating sequence is that of *sort -b*; with -t, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq*, *look*, and *awk* are wildly incongruous.

Filenames that are numeric may cause conflict when the -o option is used right before listing filenames.

4. Commands Beginning with the Letter "K"

4. kill(1)

NAME

kill - terminate a process

SYNOPSIS

```
kill [-signo] PID ...
```

DESCRIPTION

The *kill* command sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported).

Process numbers can also be found by using ps(1).

The details of the kill are described in kill(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by - is given as first argument, that signal is sent instead of terminate (see signal(2)).

In particular "kill -9 ..." is a sure kill.

SEE ALSO

csh(1), ksh(1), ps(1), sh(1), kill(2), signal(2)

4. killall(1M)

NAME

killall - cancels all active processes except the calling process

SYNOPSIS

```
killall [signal]
```

DESCRIPTION

killall is used by */etc/shutdown* to kill all active processes not directly related to the shutdown procedure.

killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

killall sends signal (see *kill(1)*) to all processes not belonging to the above group of exclusions. If no signal is specified, a default of 9 is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), shutdown(1M), kill(1), ps(1), signal(2)

WARNINGS

The killall command can be run only by the super-user.

4. ksh(1)

NAME

ksh - extended shell, the Korn shell command programming language

SYNOPSIS

```
ksh [-aefhikmnoprstuvx] [-o option] ... [-c string] [arg  ...]
```

DESCRIPTION

ksh (Korn Shell) is a command programming language that executes commands read from a terminal or a file.

4. Definitions

Metacharacter
A metacharacter is one of the following characters:

; & ( ) | < > new-line space tab

Blank
A blank is a tab or a space.

Identifier
An identifier is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for aliases, functions, and named parameters. A word is a sequence of characters separated by one or more non-quoted metacharacters.

Command
A command can be a simple command, a pipeline, or a list.

Simple Command
A simple-command is a sequence of blank separated words which may be preceded by a parameter assignment list. (See Environment below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec(2)*). The value of a simple-command is its exit status if it terminates normally, or (octal) 200+status if it terminates abnormally (see signal(2) for a list of status values).

Pipeline
A pipeline is a sequence of one or more commands separated by |. The standard output of each command but the last is connected by a *pipe(2)* to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

List
A list is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ;, &, or |&. Of these five symbols, ;, &, and |& have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish).

The symbol |& causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent Shell using the -p option of the special commands read and print described later. Only one such command can be active at any given time. The symbol && (||) causes the list following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines may appear in a list, instead of semicolons, to delimit commands.

4. Compound Commands

A command is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for identifier in word ... do list done
Each time a *for* command is executed, *identifier* is set to the next word taken from the in word list. If *in word ...* is omitted, then the *for* command executes the *do list* once for each positional parameter that is set (see Parameter Substitution below). Execution ends when there are no more words in the list.

select identifier in word ... do list done
A *select* command prints on standard error (file descriptor 2), the set of words, each preceded by a number. If *in word ...* is omitted, then the positional parameters are used instead (see Parameter Substitution below). The PS3 prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed words, then the value of the parameter *identifier* is set to the word corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the parameter identifier is set to null. The contents of the line read from standard input is saved in the parameter REPLY. The list is executed for each selection until a break or end-of-file is encountered.

case word in pattern | pattern ...) list ;; ... esac
A *case* command executes the list associated with the first pattern that matches word. The form of the patterns is the same as that used for file-name generation (see File Name Generation below).

if list then list elif list then list ... else list fi
The list following *if* is executed and, if it returns a zero exit status, the list following the first *then* is executed. Otherwise, the list following *elif* is executed and, if its value is zero, the list following the next *then* is executed. Failing that, the *else* list is executed. If no *else list* or *then list* is executed, then the *if* command returns a zero exit status.

while list do list done
until list do list done
A *while* command repeatedly executes the *while list* and, if the exit status of the last command in the list is zero, executes the *do list*; otherwise the loop terminates. If no commands in the *do list* are executed, then the *while* command returns a zero exit status; *until* may be used in place of *while* to negate the loop termination test.

(list)
Execute *list* in a separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below. A parenthesized list used as a command argument denotes process substitution as described below.

{ list;}
list is simply executed. Note that { is a keyword and requires a blank in order to be recognized.

function identifier { list ;}
identifier () { list ;}
Define a function which is referenced by identifier. The body of the function is the list of commands between { and }. (See Functions below).

time pipeline
The pipeline is executed and the elapsed time as well as the user and system time are printed on standard error.

The following keywords are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { } function select time

4. Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

4. Command Aliasing

The first word of each command is replaced by the text of an alias if an alias for this word has been defined. The first character of an alias name can be any non-special printable character, but the rest of the characters must be the same as for a valid identifier. The replacement string can contain any valid Shell script including the metacharacters listed above. The first word of each command of the replaced text will not be tested for additional aliases. If the last character of the alias value is a blank then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the keywords listed above. Aliases can be created, listed, and exported with the alias command and can be removed with the unalias command. Exported aliases remain in effect for sub-shells but must be reinitialized for separate invocations of the Shell (See Invocation below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the alias command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called tracked aliases. The value of a tracked alias is defined the first time the corresponding command is looked up and becomes undefined each time the PATH variable is reset. These aliases remain tracked so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The -h option of the set command makes each command name which is a valid alias name into a tracked alias.

The following exported aliases are compiled into the shell but can be unset or redefined:

```
false='let 0'
functions='typeset -f'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
hash='alias -t'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted tilde (~). If it does, then the word up to a / is checked to see if it matches a user name in the */etc/passwd* file. If a match is found, the tilde character and the matched login name is replaced with the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A tilde by itself, or in front of a /, is replaced by the value of the HOME parameter. A tilde followed by a + or - is replaced by the value of the parameter PWD and OLDPWD respectively.

In addition, the value of each keyword parameter is checked to see if it begins with a tilde or if a tilde appears after a colon (:). In either of these cases a tilde substitution is attempted.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign ($()) or a pair of grave accents () may be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See Quoting below). The command substitution $(cat file) can be replaced by the equivalent but faster $(<file). Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

Process Substitution

This feature is only available on versions of the support the Bull Open Software operating system that support the /dev/fd directory for naming open files. Each command argument of the form (list), <(list), or >(list) will run process list asynchronously connected to some file in /dev/fd. The name of this file will become the argument to the command. If the form with > is selected then writing on this file will provide input for list. If < is used or omitted, then the file passed as an argument will contain the output of the list process. For example,

```
paste  (cut -f1 file1)  (cut -f3 file2) | tee >(process1)  >(process2)
```

cuts fields 1 and 3 from the files file1 and file2 respectively, pastes the results together, and sends it to the processes process1 and process2, as well as putting it onto the standard output. Note that the file, which is passed as an argument to the command, is a Bull Open Software *pipe(2)*, so programs that expect to *lseek(2)* on the file will not work.

4. Parameter Substitution

A parameter is an identifier, one or more digits, or any of the characters *, @, #, ?, -, $, and !. A named parameter (a parameter denoted by an identifier) has a value and zero or more attributes. Named parameters can be assigned values and attributes by using the typeset special command. The attributes supported by the Shell are described later with the typeset special command. Exported parameters pass values and attributes to sub-shells but only values to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a subscript. A subscript is denoted by a [, followed by an arithmetic expression (see Arithmetic evaluation below) followed by a]. The value of all subscripts must be in the range of 0 through 511. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the first element.

The value of a named parameter may also be assigned by writing:

```
name=value
 ...
```

If the integer attribute, -i, is set for name the value is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, may be assigned values with the set special command. Parameter $0 is set from argument zero when the shell is invoked.

The character $ is used to introduce substitutable parameters.

${parameter}	The value, if any, of the parameter is substituted. The braces are required when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If parameter is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If parameter is * or @, then all the positional parameters, starting with $1, are substituted (separated by a field separator character). If an array identifier with subscript * or @ is used, then the value for each of the elements is substituted (separated by a field separator character).
${#parameter}	If parameter is * or @, the number of positional parameters is substituted. Otherwise, the length of the value of the parameter is substituted.
${#identifier[*]}	The number of elements in the array identifier is substituted.
${parameter:-word}	If parameter is set and is non-null then substitute its value; otherwise substitute word.
${parameter:=word}	If parameter is not set or is null then set it to word; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.
${parameter:?word}	If parameter is set and is non-null then substitute its value; otherwise, print word and exit from the shell. If word is omitted then a standard message is printed.
${parameter:+word}	If parameter is set and is non-null then substitute word; otherwise substitute nothing.

${parameter#pattern}
${parameter##pattern} If the Shell pattern matches the beginning of the value of parameter, then the value of this substitution is the value of the parameter with the matched portion deleted; otherwise the value of this parameter is substituted. In the first form the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

${parameter%pattern}
${parameter%%pattern} If the Shell pattern matches the end of the value of parameter, then the value of parameter with the matched part deleted; otherwise substitute the value of parameter. In the first form the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

In the above, word is not evaluated unless it is to be used as the substituted string, so that, in the following example, pwd is executed only if d is not set or is null:

```
echo ${d:-$(pwd)}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether parameter is set or not.

4. Predefined Parameters

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the set command.
?	The decimal value returned by the last executed command.
$	The process number of this shell.
_	(underscore) The last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching MAIL file when checking for mail. Finally, the value of this parameter is set to the full path name of each program the shell invokes and is passed in the environment.
!	The process number of the last background command invoked.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the cd command.
OLDPWD	The previous working directory set by the cd command.

RANDOM	Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric value to RANDOM.
REPLY	This parameter is set by the select statement and by the read special command when no arguments are supplied.
SECONDS	Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

4. Variables Used by the Shell

The following parameters are used by the shell:

CDPATH	The search path for the cd command.
COLUMNS	If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing select lists.
EDITOR	If the value of this variable ends in emacs, gmacs, or vi and the VISUAL variable is not set, then the corresponding option (see Special Command set below) will be turned on.
ENV	If this parameter is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the shell is invoked. (See Invocation below.) This file is typically used for alias and function definitions.
FCEDIT	The default editor name for the fc command.
IFS	Internal field separators, normally space, tab, and new-line that is used to separate command words which result from command or parameter substitution and for separating words with the special command read. The first character of the IFS parameter is used to separate arguments for the "$*" substitution (See Quoting below).
HISTFILE	If this parameter is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See Command re-entry below.)
HISTSIZE	If this parameter is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

HOME	The default argument (home directory) for the cd command.
LINES	If this variable is set, the value is used to determine the column length for printing select lists. Select lists will print vertically until about two-thirds of LINES lines are filled.
MAIL	If this parameter is set to the name of a mail file and the MAILPATH parameter is not set, then the shell informs the user of arrival of mail in the specified file.
MAILCHECK	This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the MAILPATH or MAIL parameters. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.
MAILPATH	A colon (:) separated list of file names. If this parameter is set then the shell informs the user of any modifications to the specified files that have occurred within the last MAILCHECK seconds. Each file name can be followed by a ? and a message that will be printed. The message will undergo parameter and command substitution with the parameter, $_ defined as the name of the file that has changed. The default message is you have mail in $_.
PATH	The search path for commands (see Execution below). The user may not change PATH if executing under rsh (except in .profile).
PS1	The value of this parameter is expanded for parameter substitution to define the primary prompt string which by default is ``$ ''. The character ! in the primary prompt string is replaced by the command number (see Command Re-entry below).
PS2	Secondary prompt string, by default ``> ''.
PS3	Selection prompt string used within a select loop, by default ``#? ''.
SHELL	The pathname of the shell is kept in the environment. At invocation, if the value of this variable contains an r in the basename, then the shell becomes restricted.
TMOUT	If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the PS1 prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)
VISUAL	If the value of this variable ends in emacs, gmacs, or vi then the corresponding option (see Special Command set below) will be turned on.

The shell gives default values to PATH, PS1, PS2, MAILCHECK, TMOUT and IFS, while HOME, SHELL ENV and MAIL are not set at all by the shell (although HOME is set by login(1)). On some systems MAIL and SHELL are also set by login(1)).

4. Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

4. File Name Generation

Following substitution, each command word is scanned for the characters * (asterisk), ? (question mark), and [(left bracket) unless the -f option has been set. If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged.

When a pattern is used for file name generation, the character . (dot) at the start of a file name or immediately following a / (slash), as well as the character / itself, must be matched explicitly. In other instances of pattern matching the / (slash) and . (dot) are not treated specially.

* Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "! " then any character not enclosed is matched. A - can be included in the character set by putting it as the first or last character.

4. Quoting

Each of the metacharacters listed above (See Definitions above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be quoted (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks ('), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, , ", and $. The meaning of $* and $@ is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, "$*" is equivalent to "$1d$2d...", where d is the first character of the IFS parameter, whereas "$@" is equivalent to "$1" "$2" ....

Inside grave quote marks () \ quotes the characters \, ` (grave accent), and $. If the grave quotes occur within double quotes then \ also quotes the character ".

The special meaning of keywords or aliases can be removed by quoting any character of the keyword. The recognition of function names or special command names listed below cannot be altered by quoting them.

4. Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command let. Evaluations are performed using long arithmetic. Constants are of the form base#n where base is a decimal number between two and thirty-six representing the arithmetic base and n is a number in that base. If base is omitted then base 10 is used.

An internal integer representation of a named parameter can be specified with the -i option of the typeset special command. When this attribute is selected the first assignment to the parameter determines the arithmetic base to be used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the let command is provided. For any command which begins with a ((, all the characters until a matching)) are treated as a quoted expression. More precisely, ((...)) is equivalent to let "...".

4. Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of PS2) is issued.

4. Input and Output Redirection

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command. Command and parameter substitution occurs before word or digit is used except as noted below.

File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<word	Use file word as standard input (file descriptor 0).
>word	Use file word as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
>>word	Use file word as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<-word	The shell input is read up to a line that is the same as word, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on word. The resulting document, called a here-document, becomes the standard input. If any character of word is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, \new-line is ignored, and \ must be used to quote the characters \, $, , and the first character of word. If - is appended to <<, then all leading tabs are stripped from word and from the document.
<&digit	The standard input is duplicated from file descriptor digit (see dup(2)). Similarly for the standard output using >& digit.
<&-	The standard input is closed. Similarly for the standard output using >&-.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1. The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (file descriptor, file) association at the time of evaluation. For example:

```
... 1>fname 2>&1
```

first associates file descriptor 1 with file fname. It then associates file descriptor 2 with the file associated with file descriptor 1 (i.e. fname). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file fname.

If a command is followed by & and job control is not active, then the default standard input for the command is the empty file /dev/null. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

4. Environment

The environment (see *environ(7)*) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be identifiers and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value and marking it export . Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the export or typeset -x commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in export or typeset -x commands.

The environment for any simple-command or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word of the form identifier=value. Thus, the two following examples are equivalent (as far as the above execution of cmd is concerned):

```
TERM=hw10 cmd args
(export TERM; TERM=hw10; cmd args)
```

If the -k flag is set, all parameter assignment arguments are placed in the environment, even if they occur after the command name. The following first prints a=b c and then c:

```
echo a=b c
set -k
echo a=b c
```

4. Functions

The function keyword, described in the Commands section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See Execution below).

Functions execute in the same process as the caller and share all files, traps (other than EXIT and ERR) and present working directory with the caller. A trap set on EXIT inside a function is executed after the function completes. Ordinarily, variables are shared between the calling program and the function. However, the typeset special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command return is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the -f option of the typeset special command. The text of functions will also be listed. Function can be undefined with the -f option of the unset special command.

Ordinarily, functions are unset when the shell executes a shell script. The -xf option of the typeset command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be placed in the ENV file.

4. Job Control

If the monitor option of the set command is turned on, an interactive shell associates a job with each pipeline. It keeps a table of current jobs, printed by the jobs command, and assigns them small integer numbers. When a job is started asynchronously with &, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the bg command, or run some other commands and then eventually bring the job back into the foreground with the foreground command fg. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command "stty tostop". If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character % introduces a job name. If you wish to refer to job number 1, you can name it as %1 . Jobs can also be named by prefixes of the string typed in to kill or restart them.

Thus, on systems that support job control, `fg %ed' would normally restart a suspended *ed(1)* job, if there were a suspended job whose name began with the string 'ed'.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a -. The abbreviation %+ refers to the current job and %- refers to the previous job. %% is also a synonym for the current job.

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) jobs.' You may use the jobs command to see what they are.

If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

4. Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by & and job monitor option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the trap command below).

4. Command Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the Special Commands listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the function call. When the function completes or issues a return, the positional parameter list is restored and any trap set on EXIT within the function is executed.

The value of a function is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a special command or a user defined function, a process is created and an attempt is made to execute the command via *exec(2)*.

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is /bin:/usr/bin: (specifying /bin, /usr/bin, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list.

If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an a.out file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and named parameters are removed in this case. If the shell command file doesn't have read permission, or if the setuid and/or setgid bits are set on the file, then the shell executes an agent whose job it is to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is also executed in a sub-shell without removing non-exported quantities.

4. Command Re-entry

The text of the last HISTSIZE (default 128) commands entered from a terminal device is saved in a history file. The file $HOME/.sh_history is used if the HISTFILE variable is not set or is not writable. A shell can access the commands of all interactive shells which use the same named HISTFILE. The special command fc is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to fc then the value of the parameter FCEDIT is used. If FCEDIT is not defined then /bin/ed is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name - is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form old=new can be used to modify the command before execution. For example, if r is aliased to 'fc -e -' then typing `r bad=good c' will re- execute the most recent command which starts with the letter c, replacing the first occurrence of the string bad with the string good.

4. In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a new-line (RETURN or LINE FEED). If either the emacs, gmacs, or vi option is active, the user can edit the command line. To be in either of these edit modes set the corresponding option. An editing option is automatically selected each time the VISUAL or EDITOR variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept RETURN as carriage return without line feed and that a space must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to "space".

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of COLUMNS if it is defined, otherwise 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user.

As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window.

Emacs Editing Mode

This mode is entered by enabling either the emacs or gmacs option. The only difference between these two modes is the way they handle ^T. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character.

For example, ^F is the notation for control F. This is entered by depressing f while holding down the CTRL (control) key. The SHIFT key is not depressed. (The notation ^? indicates the DEL (delete) key.)

The notation for escape sequences is M- followed by a character. For example, M-f (pronounced Meta f) is entered by depressing ESC (ascii 033) followed by f. (M-F would be the notation for ESC followed by SHIFT (capital) F.)

All edit commands operate from any place on the line (not just at the beginning). Neither the RETURN nor the LINE FEED key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.

^]char	Move cursor to character char on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the stty command, usually ^H or #.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is ^? (DEL, the default) then this command will not work).
^T	Transpose current character with next character in emacs mode. Transpose two previous characters in gmacs mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lower case.
^K	Kill from the cursor to the end of the line. If given a parameter of zero then kill from the start of line to the cursor.
^W	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User defined kill character as defined by the stty command, usually ^G or @.) Kill the entire current line. If two kill characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(Null character) Set mark.
M-	(Meta space) Set mark.
^J	(New line) Execute the current line.
^M	(Return) Execute the current line.

eof	End-of-file character, normally ^D, will terminate the shell if the current line is null.
^P	Fetch previous command. Each time ^P is entered the previous command back in time is accessed.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
^N	Fetch next command. Each time ^N is entered the next command forward in time is accessed.
^Rstring	Reverse search history for a previous command line containing string. If a parameter of zero is given, the search is forward. String is terminated by a "RETURN" or "NEW LINE". If string is omitted, then the next command line containing the most recent string is accessed. In this case a parameter of zero reverses the direction of the search.
^O	Operate - Execute the current line and fetch the next line relative to current line from the history file.
M-digits	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are ., ^F, ^B, erase, ^D, ^K, ^R, ^P, ^N, M-., M-_, M-b, M-c, M- d, M-f, M-h and M-^H.
M-letter	Soft-key - Your alias list is searched for an alias by the name _letter and if an alias of this name is defined, its value will be inserted on the input queue. The letter must not be one of the above meta-functions.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as M-..
M-*	Attempt file name generation on the current word. An asterisk is appended if the word doesn't contain any special pattern characters.
M-ESC	Same as M-*.
M-=	List files matching current word pattern if an asterisk were appended.
^U	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill and interrupt (normally ^?) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
^V	Display version of the shell.

Vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the input mode. To edit, the user enters control mode by typing ESC (033) and moves the cursor

to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat count prior to the command.

When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option *viraw* is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	(User defined erase character as defined by the stty command, usually ^H or #.) Delete previous character.
^W	Delete the previous blank separated word.
^D	Terminate the shell.
^V	Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a ^V. The ^V removes the next character's editing features (if any).
\	Escape the next erase or kill character.

Motion Edit Commands

These commands will move the cursor.

[count]l	Cursor forward (right) one character.
[count]w	Cursor forward one alpha-numeric word.
[count]W	Cursor to the beginning of the next word that follows a blank.
[count]e	Cursor to end of word.
[count]E	Cursor to end of the current blank delimited word.
[count]h	Cursor backward (left) one character.

[count]b	Cursor backward one word.
[count]B	Cursor to preceding blank separated word.
[count]fc	Find the next character c in the current line.
[count]Fc	Find the previous character c in the current line.
[count]tc	Equivalent to f followed by h.
[count]Tc	Equivalent to F followed by l.
;	Repeats the last single character find command, f, F, t, or T.
,	Reverses the last single character find command.
0	Cursor to start of line.
^	Cursor to first non-blank character in line.
$	Cursor to end of line.

Search Edit Commands

These commands access your command history.

[count]k	Fetch previous command. Each time k is entered the previous command back in time is accessed.
[count]-	Equivalent to k.
[count]j	Fetch next command. Each time j is entered the next command forward in time is accessed.
[count]+	Equivalent to j.
[count]G	The command number count is fetched. The default is the least recent history command.
/string	Search backward through history for a previous command containing string. String is terminated by a RETURN or NEW LINE. If string is null the previous string will be used.
?string	Same as / except that search will be in the forward direction.
n	Search for next match of the last pattern to / or ? commands.
N	Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the string entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a	Enter input mode and enter text after the current character.
A	Append text to the end of the line. Equivalent to $a.
[count]cmotion c[count]motion	Delete current character through the character that motion would move the cursor to and enter input mode. If motion is c, the entire line will be deleted and input mode entered.
C	Delete the current character through the end of line and enter input mode. Equivalent to c$.
S	Equivalent to cc.
D	Delete the current character through the end of line. Equivalent to d$.
[count]dmotion d[count]motion	Delete current character through the character that motion would move to. If motion is d, the entire line will be deleted.
i	Enter input mode and insert text before the current character.
I	Insert text before the beginning of the line. Equivalent to the two character sequence ^i.
[count]P	Place the previous text modification before the cursor.
[count]p	Place the previous text modification after the cursor.
R	Enter input mode and replace characters on the screen with characters you type overlay fashion.
rc	Replace the current character with c.
[count]x	Delete current character.
[count]X	Delete preceding character.
[count].	Repeat the previous text modification command.
~	Invert the case of the current character and advance the cursor.
[count]_	Causes the count word of the previous command to be appended and input mode entered. The last word is used if count is omitted.
*	Causes an * to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

Miscellaneous Edit Commands

[count]ymotion y[count]motion	Yank current character through character that motion would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.
Y	Yanks from current position to end of line. Equivalent to y$.
u	Undo the last text modifying command.
U	Undo all the text modifying commands performed on the line.
[count]v	Returns the command fc -e
${VISUAL:-${EDITOR:-vi}}	Count in the input buffer. If count is omitted, then the current line is used.
^L	Line feed and print current line. Has effect only in control mode.
^J	(New line) Execute the current line, regardless of mode.
^M	(Return) Execute the current line, regardless of mode.
#	Sends the line after inserting a # in front of the line and after each new-line. Useful for causing the current line to be inserted in the history without being executed.
=	List the filenames that match the current word if an asterisk were appended it.
@letter	Your alias list is searched for an alias by the name _letter and if an alias of this name is defined, its value will be inserted on the input queue for processing.

4. Special Commands

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1.

Commands that are preceded by one or two - are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.

2. They are executed in a separate process when used within command substitution.

3. Errors in commands preceded by -- cause the script that contains them to abort.

- : arg ...
The command only expands parameters. A zero exit code is returned.

-- . file arg ...
Read and execute commands from file and return. The commands are executed in the current Shell environment. The search path specified by PATH is used to find the directory containing file. If any arguments arg are given, they become the positional parameters. Otherwise the positional parameters are unchanged.

alias -tx name =value ...
Alias with no arguments prints the list of aliases in the form name=value on standard output. An alias is defined for each name whose value is given. A trailing space in value causes the next word to be checked for alias substitution. The -t flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given name.

The value becomes undefined when the value of PATH is reset but the aliases remained tracked. Without the -t flag, for each name in the argument list for which no value is given, the name and value of the alias is printed. The -x flag is used to set or print exported aliases. An exported alias is defined across sub-shell environments. Alias returns true unless a name is given for which no alias has been defined.

bg %job
This command is only built-in on systems that support job control. Puts the specified job into the background. The current job is put in the background if job is not specified.

break n
Exit from the enclosing for while until or select loop, if any. If n is specified then break n levels.

continue n
Resume the next iteration of the enclosing for while until or select loop. If n is specified then resume at the n-th enclosing loop.

- cd arg
- cd old new
This command can be in either of two forms. In the first form it changes the current directory to arg. If arg is - the directory is changed to the previous directory. The shell parameter HOME is the default arg. The parameter PWD is set to the current directory. The shell parameter CDPATH defines the search path for the directory containing arg.

Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If arg begins with a / then the search path is not used. Otherwise, each directory in the path is searched for arg.

The second form of cd substitutes the string new for the string old in the current directory name, PWD and tries to change to this new directory.

The cd command may not be executed by rsh.

echo arg ...
See echo(1) for usage and description.

-- eval arg ...
The arguments are read as input to the shell and the resulting command(s) executed.

-- exec arg ...
If arg is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

exit n
Causes the shell to exit with the exit status specified by n. If n is omitted then the exit status is that of the last command executed. An end-of-file will also cause the shell to exit except for a shell which has the ignoreeof option (See set below) turned on.

-- export name ...
The given names are marked for automatic export to the environment of subsequently-executed commands.

-- fc -e ename -nlr first last
-- fc -e - old=new command
In the first form, a range of commands from *first* to *last* is selected from the last HISTSIZE commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number.

If the flag -l, is selected, the commands are listed on standard output. Otherwise, the editor program ename is invoked on a file containing these keyboard commands. If ename is not supplied, then the value of the parameter FCEDIT (default /bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed.

If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and -16 for listing. The flag -r reverses the order of the commands and the flag -n suppresses command numbers when listing. In the second form the command is re- executed after the substitution old=new is performed.

fg %job
This command is only built-in on systems that support job control. If job is specified it brings it to the foreground. Otherwise, the current job is brought into the foreground.

jobs -l
Lists the active jobs; given the -l options lists process id's in addition to the normal information.

kill -sig process ...
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes.

Signals are either given by number or by names (as given in <signal.h>, stripped of the prefix "SIG").

The signal numbers and names are listed by 'kill -l'.

If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument process can be either a process id or a job.

let arg ...
Each arg is an arithmetic expression to be evaluated. All calculations are done as long integers and no check for overflow is performed. Expressions consist of constants, named parameters, and operators. The following set of operators, listed in order of decreasing precedence, have been implemented:

-	unary minus
!	logical negation
* / %	multiplication, division, remainder
+ -	addition, subtraction
<= >= < >	comparison
== !=	equality inequality
=	arithmetic replacement

Sub-expressions in parentheses () are evaluated first and can be used to override the above precedence rules. The evaluation within a precedence group is from right to left for the = operator and from left to right for the others.

A parameter name must be a valid identifier. When a parameter is encountered, the value associated with the parameter name is substituted and expression evaluation resumes. Up to nine levels of recursion are permitted.

The return code is 0 if the value of the last expression is non-zero, and 1 otherwise.

print -Rnprsun arg ...
The shell output mechanism. With no flags or with flag -, the arguments are printed on standard output as described by echo(1). In raw mode, -R or -r, the escape conventions of echo are ignored. The -R option will print all subsequent arguments and options other than -n. The -p option causes the arguments to be written onto the pipe of the process spawned with |& instead of standard output. The -s option causes the arguments to be written onto the history file instead of standard output. The -u flag can be used to specify a one digit file descriptor unit number n on which the output will be placed. The default is 1. If the flag -n is used, no new-line is added to the output.

pwd
Equivalent to print -r - $PWD

read -prsu n name?prompt name ...
The shell input mechanism. One line is read and is broken up into words using the characters in IFS as separators. In raw mode, -r, a \ at the end of a line does not signify line continuation. The first word is assigned to the first name, the second word to the second name, etc., with leftover words assigned to the last name. The -p option causes the input line to be taken from the input pipe of a process spawned by the shell using |&. If the -s flag is present, the input will be saved as a command in the history file. The flag -u can be used to specify a one digit file descriptor unit to read from. The file descriptor can be opened with the exec special command.

The default value of n is 0. If name is omitted then REPLY is used as the default name. The return code is 0 unless an end-of-file is encountered. An end-of-file with the -p option causes cleanup for this process so that another can be spawned. If the first argument contains a ?, the remainder of this word is used as a prompt when the shell is interactive. If the given file descriptor is open for writing and is a terminal device then the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2.

The return code is 0 unless an end-of-file is encountered.

-- readonly name ...
The given names are marked readonly and these names cannot be changed by subsequent assignment.

-- return n
Causes a shell function to return to the invoking script with the return status specified by n. If n is omitted then the return status is that of the last command executed. If return is invoked while not in a function or a script, then it is the same as an exit.

set -aefhkmnostuvx -o option ... arg ...
The flags for this command have meaning as follows:

-a All subsequent parameters that are defined are automatically exported.

-e If the shell is non-interactive and if a command fails, execute the ERR trap, if set, and exit immediately. This mode is disabled while reading profiles.

-f Disables file name generation.

-h Each command whose name is an identifier becomes a tracked alias when first encountered.

-k All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.

-m Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

-n Read commands but do not execute them. Ignored for interactive shells.

-o The following argument can be one of the following option names:

allexport	Same as -a.
errexit	Same as -e.
bgnice	All background jobs are run at a lower priority.
emacs	Puts you in an emacs style in-line editor for command entry.
gmacs	Puts you in a gmacs style in-line editor for command entry.
ignoreeof	The shell will not exit on end-of-file. The command exit must be used.
keyword	Same as -k.
markdirs	All directory names resulting from file name generation have a trailing / appended.
monitor	Same as -m.
noexec	Same as -n.
noglob	Same as -f.
nounset	Same as -u.
protected	Same as -p.
verbose	Same as -v.
trackall	Same as -h.
vi	Puts you in insert mode of a vi style in-line editor until you hit escape character 033. This puts you in move mode. A return sends the line.
viraw	Each character is processed as it is typed in vi mode.
xtrace	Same as -x. If no option name is supplied then the current option settings are printed.
-p	Resets the PATH variable to the default value, disables processing of the $HOME/.profile file and uses the file /etc/suid_profile instead of the ENV file. This mode is automatically enabled whenever the effective uid (gid) is not equal to the real uid (gid).
-s	Sort the positional parameters.
-t	Exit after reading and executing one command.
-u	Treat unset parameters as an error when substituting.
-v	Print shell input lines as they are read.
-x	Print commands and their arguments as they are executed.
-	Turns off -x and -v flags and stops examining arguments for flags.
--	Do not change any of the flags; useful in setting $1 to a value beginning with -. If no arguments follow this flag then the positional parameters are unset.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $-. The remaining arguments are positional parameters and are assigned, in order, to $1 $2 .... If no arguments are given then the values of all names are printed on the standard output.

- shift n

The positional parameters from $n+1 ... are renamed $1 ..., default n is 1. The parameter n can be any arithmetic expression that evaluates to a non-negative number less than or equal to $#.

test expr
Evaluate conditional expression expr. See *test(1)* for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional primitive expressions are allowed:

-L file	True if file is a symbolic link.
file1 -nt file2	True if file1 is newer than file2.
file1 -ot file2	True if file1 is older than file2.
file1 -ef file2	True if file1 has the same device and i-node number as file2.

times
Print the accumulated user and system times for the shell and for processes run from the shell.

trap arg sig ...
arg is a command to be read and executed when the shell receives signal(s) sig. (Note that arg is scanned once when the trap is set and once when the trap is taken.)

Each sig can be given as a number or as the name of the signal. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective.

If arg is omitted or is -, then all trap(s) sig are reset to their original values. If arg is the null string then this signal is ignored by the shell and by the commands it invokes. If sig is ERR then arg will be executed whenever a command has a non-zero exit code. This trap is not inherited by functions.

If sig is 0 or EXIT and the trap statement is executed inside the body of a function, then the command arg is executed after the function completes. If sig is 0 or EXIT for a trap set outside any function then the command arg is executed on exit from the shell. The trap command with no arguments prints a list of commands associated with each signal number.

-- typeset -HLRZfilprtuxn name =value ...
When invoked inside a function, a new instance of the parameter name is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

-H This flag provides Bull Open Software to host-name file mapping on non-Bull Open Software machines.

-L Left justify and remove leading blanks from value.

If n is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. When the parameter is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the -Z flag is also set. The -R flag is turned off.

-R Right justify and fill with leading blanks. If n is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the parameter is reassigned. The L flag is turned off.

-Z Right justify and fill with leading zeros if the first non-blank character is a digit and the -L flag has not been set. If n is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment.

-f The names refer to function names rather than parameter names. No assignments can be made and the only other valid flags are -t, which turns on execution tracing for this function and -x, to allow the function to remain in effect across shell procedures executed in the same process environment.

-i Parameter is an integer. This makes arithmetic faster. If n is non-zero it defines the output arithmetic base, otherwise the first assignment determines the output base.

-l All upper-case characters converted to lower-case. The upper-case flag, -u is turned off.

-p The output of this command, if any, is written onto the two-way pipe

-r The given names are marked readonly and these names cannot be changed by subsequent assignment.

-t Tags the named parameters. Tags are user definable and have no special meaning to the shell.

-u All lower-case characters are converted to upper-case characters. The lower-case flag, -l is turned off.

-x The given names are marked for automatic export to the environment of subsequently-executed commands.

Using + rather than - causes these flags to be turned off. If no name arguments are given but flags are specified, a list of names (and optionally the values) of the parameters which have these flags set is printed. (Using + rather than - keeps the values to be printed.) If no names and flags are given, the names and attributes of all parameters are printed.

ulimit -acdfmpst n
If no option is given, -f is assumed. If n is not given the current limit is printed.

-a Lists all of the current resource limits.

-c imposes a size limit of n 512 byte blocks on the size of core dumps.

-d imposes a size limit of n kbytes on the size of the data area.

-f imposes a size limit of n 512 byte blocks on files written by child processes (files of any size may be read).

-m imposes a soft limit of n kbytes on the size of physical memory.

-p changes the pipe size to n.

-s imposes a size limit of n kbytes on the size of the stack area.

-t imposes a time limit of n seconds to be used by each process.

umask nnn
The user file-creation mask is set to nnn (see umask(2)). If nnn is omitted, the current value of the mask is printed.

unalias name ...
The parameters given by the list of names are removed from the alias list.

unset -f name ...
The parameters given by the list of names are unassigned, i. e., their values and attributes are erased. Readonly variables cannot be unset. If the flag, -f, is set, then the names refer to function names.

wait n
Wait for the specified child process and report its termination status. If n is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

whence -v name ...
For each name, indicate how it would be interpreted if used as a command name. The flag, -v, produces a more verbose report.

4. Invocation

If the shell is invoked by *exec(2)*, and the first character of argument zero ($0) is -, then the shell is assumed to be a login shell and commands are read from /etc/profile and then from either .profile in the current directory or $HOME/.profile, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter ENV if the file exists.

If the -s flag is not present and arg is present, then a path search is performed on the first arg to determine the name of the script to execute. The script arg must have read permission and any setuid and getgid settings will be ignored. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

-c string	If the -c flag is present then commands are read from string.
-s	If the -s flag is present or if no arguments remain then commands are read from the standard input. Shell output, except for the output of the Special commands listed above, is written to file descriptor 2.
-i	If the -i flag is present or if the shell input and output are attached to a terminal (as told by ioctl(2)) then this shell is interactive. In this case TERM is ignored (so that kill 0 does not kill an interactive shell) and INTR is caught and ignored (so that wait is interruptible). In all cases, QUIT is ignored by the shell.
-r	If the -r flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the set command above.

4. EXIT STATUS, FILES, SEE ALSO, BUGS

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the exit command above). If the shell is being used non-interactively then execution of the shell file is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets ([]) after the command or function name.

FILES

/etc/passwd
/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cat(1), cd(1), echo(1), env(1), vi(1), a.out(4).

emacs(1), gmacs(1), test(1), umask(1), in the *User's Reference Manual*.

dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), signal(2), umask(2), ulimit(2), wait(2), rand(3), profile(4), environ(5) in the *Programmer's Reference Manual*.

BUGS

If a command which is a tracked alias is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to exec the original command. Use the -t option of the alias command to correct this situation.

Some very old shell scripts contain a ^ as a synonym for the pipe character |.

If a command is piped into a shell command, then all variables set in the shell command are lost when the command completes.

Using the *fc* built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command *. file* reads the whole file before any commands are executed. Therefore, alias and unalias commands in the file will not apply to any functions defined in the file.

5. Commands Beginning with the Letter "L"

5. labelit(1M)

Copy file systems with label checking. Referred to in volcopy(1M).

5. lanspy

NAME

lanspy - OPEN 7 tool to trace the communications between OPEN 7 and a remote UNIX system

SYNOPSIS

```
lanspy [-a] [-b num] [-c eaddr] [-d format] [-e num] [-f] [-h]
       [-i ifile] [-m size] [-o ofile] [-r header] [-s]
       [-w delay] [-x string]
       [-A] [-C protocol] [-D] [-E eaddr] [-F delay] [-I protocol]
       [-M size] [-N iaddr] [-P] [-R] [-S] [-T tport] [-U uport]
       [-X string]
```

DESCRIPTION

OPEN 7 implements TCP/IP on Ethernet, FDDI, and X25 (with the AX25 product on DNS, CNS ...). Here we focus on the most usual links: FDDI or Ethernet.

The *lanspy* command implements the OPEN 7 trace. It is used to find whether a problem occurs in the OPEN 7 side of the link, or in the remote UNIX system side. *lanspy* starts, stops and edits the trace. The main options are shown below; all options are listed under the OPTIONS section.

```
lanspy -o ofile                    (1)
lanspy -D                          (2)
lanspy -f                          (3)
lanspy -i ifile                    (4)
```

(1) This command enables network spy and write frames in *ofile*. It is used to collect the traces.

(2) This command collects headers and data fields (default is headers only). It option is mandatory to get the data transferred.

(3) This command disable network spy. It is used to stop the collect.

(4) This command reads frames from *ifile*. It is used to analyse the collect.

This trace is very heavy, so try to use it for reproductible problems.

You can use a separate file system to avoid saturation of a file system.

OPTIONS

All options are described below:

-a	automatic frames writing mode.
-b num	Begin display at frame number *num*.
-c eaddr	collect only frames with this Ethernet address.
-d format	Display data with the following format: no do not display data asc display data in ASCII format hex display data in hexadecimal + ASCII format full display all data in frame
-e num	End display at frame number *num*.
-f	disable network spy. Use this option to stop the collect.
-h	Display the help (the present list of options).
-i ifile	read frames from *ifile*. Use this option to analyse the collect.
-m size	Display only first *size* bytes of data.
-o ofile	enable network spy and write frames in *ofile*. This option is used to collect the traces.
-r header	Set raw mode for this *header*. Allowed *header* values are: eth, arp, pup, ip, icmp, ggp, tcp, egp, ipup, udp, idp, all.
-s	Give current status of network spy.
-w delay	collect frames while *delay* seconds
-x string	Prefix input frames with *string*.
-A	Do not display ARP Ethernet frames.
-C protocol	collect only frames with this protocol. This option is sometimes used to get a selective collect. Allowed protocol values are: arp, pup, raw, icmp, ggp, tcp, egp, ipup, udp, idp, ipall.
-D	collect headers and data fields (default is headers only).This option is mandatory to get the data transferred.
-E eaddr	Display only Ethernet frames with address *eaddr*.
-F delay	Flush the network spy buffer every *delay* seconds.
-I protocol	Do not display IP Ethernet frames for this *protocol*. Allowed *protocol* values are: raw, icmp, ggp, tcp, egp, ipup, udp, idp, all.

-M size	Display only last *size* bytes of data.
-N iaddr	Display only internet frames with address *iaddr*.
-P	Do not display PUP Ethernet frames.
-R	Display all frames in raw mode.
-S	Print size of headers
-T tport	Display only TCP frames with port number *tport*.
-U uport	Display only UDP frames with port number *uport*.
-X string	Prefix output frames with *string*.

EXAMPLES

The following example is not intended to explain TCP/IP, but to give a practical example of trace, and to point out some important details.

```
CLIENT                                                  SERVER

                                                        LISTEN
opening the connection -------- SYNj --------------->
SYN_RCVD

ESTABLISHED            <------- SYNk, ACKj+1 --------

ESTABLISHED            -------- ACKk+1 ------------->

                       ... Exchanging data ...

This side closes
the connection

FIN_WAIT_1             -------- FINm --------------->
CLOSE_WAIT

FIN_WAIT_1             <-------- ACKm+1 -------------

TIME_WAIT              <-------- FINn -------------->
LAST_ACK
                       -------- ACKn+1 ------------->

CLOSED
```

All these states of a connection can be seen by the command *netstat -a*.

First: an example of a good ftp connection between BMA5 (open7 2.0.3) and cs7 (dpx20 3.2.5)

```
# arp -a
pagot (129.185.214.80) at 0:20:9:0:1f:93
```

Start the trace
==========

```
# lanspy -o /tmp/toto -D
lanspy: network spy on file /tmp/toto

#ftp cs7
Connected to cs7
220 cs7 FTP server (Version 4.9 Thu Sep2 20:35:07 CDT 1993) ready
Name (cs7:root):rama
331 Password required for rama
Password:
230 User rama logged in
ftp>pwd
257 "/home/rama" is current directory.
ftp> ls
200 PORT command successful
150 Opening data connection for /bin/ls.
cngtw
ftp
po7win1.exe
po7win10.exe
.....
226 Transfer complete;
220 bytes received in 0.016 seconds (13 Kbytes/s)
ftp>quit
221 Goodbye.
```

Stop the Trace
==========

```
# lanspy -f
lanspy: network accounting stopped
#lanspy: network accounting stopped
```

Edit the Trace
==========

```
#lanspy -i /tmp/toto> /tmp/toto-edt


# arp -a
pagot (129.185.214.80) at 0:20:9:0:1f:93
cs7 (129.185.214.52) at 8:0:5a:1:c9:7a

#view /tmp/toto-edt
```

Only the traffic relative to the ftp session is selected.

In the edited trace, the name of the systems are edited instead of the IP address, according to correspondance in */etc/hosts* (otherwise you have only the IP address). It is the same for the port edited according to */etc/services*.

First the ARP protocol is used to resolve the Ethernet address of cs7.

Connection Identification in TCP/IP

A connection is identified by the IP address and the port number of the source and the IP address and the port number of the destination.

In this frame, BMA5 sends a broadcast to ask cs7 its Ethernet address.

```
220: time=9:6:54:916 [35] [13887] length=64
ether : 8:0:38:50:64:a5 --> ff:ff:ff:ff:ff:ff (EBROADCAST) TYPE_ARP
arp   : hardware: format=ETHER length=6 protocol: format=IP length=4 REQUEST
arp   : 8:0:38:50:64:a5 --> 0:0:0:0:0:0 (EBROADCAST)
arp   : 129.185.216.36 (bma5) --> 129.185.216.54 (cs7)
```

cs7 replies to bma5, giving its Ethernet address

```
221: time=9:6:54:922 [6] [13893] length=64
```

5. ld(1)

NAME

ld - the OPEN 7 linker for common object files

SYNOPSIS

```
ld [options] filename
```

DESCRIPTION

The *ld* command combines several object files into one, relocates object files, resolves external symbol references, and supports symbol table information for symbolic debugging.

In the simplest case, the names of several object programs are given, and *ld* combines them, producing an object module which can either be executed, or used as input for a subsequent run of *ld*. The output of *ld* is placed in *a.out*. By default, this file is executable if no errors occurred during the link.

If an input file is not an object file, *ld* assumes that it is either an archive library, or a text file containing link editor directives (see the Link Editor chapter in the *UNIX System V Programmer's Guide* for a discussion of input directives.)

If an argument is a library, it is searched just once at the point it is encountered in the argument list. Only those routines which define unresolved external references are loaded. The library (archive) symbol table (see *ar*(4)) is searched sequentially with as many passes as are neccessary to resolve external references which can be satisfied by library members. Thus the ordering of library members is unimportant.

The following options are recognized by *ld*:

-e epsym	Set the default entry point address for the output file to be that of the symbol *epsym*.
-f fill	Set the default fill pattern for "holes" within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.
-lx	Search a library *libx.a*, where x is up to nine characters. A library is searched when its name is encountered, so the position of ***-l*** is significant. By default, libraries are located in */lib* and */usr/lib*.
-m	Produce a map of the input/output sections in the standard output file.

-o outfile — Produce an output object file with the name *outfile*. The name of the default object file is with *a.out*.

-r — Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor does not report unresolved references, and the output file is not executed.

-s — Strip the line number entries and symbol table information from the output object file.

-t — Turn off the warning about multiple-defined symbols that are not of the same size.

-u symname — Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-x — Do not preserve local (non *.glob*) symbols in the output symbol table, enters external and static symbols only. This option saves some space in the output file.

-L dir — Change the search algorithm so that *libx.a* looks in *dir* before looking in */lib* and */usr/lib*. This option is effective only if it precedes the ***-l*** option on the command line.

-T — Put the heap in a large segment, 4 Mbytes. Default is small segment, 64 Kbytes.

-P — Put the stack in a large segment, 4 Mbytes. Default is small segment, 64 Kbytes.

-V — Output a message giving information about the version of ld being used.

FILES

```
/lib/crt0.o      startup module
/lib/lib*.a      libraries
/usr/lib/lib*.a  libraries
a.out            output file
```

APPLICATION NOTES

Through its options and input directives, the common link editor provides great flexibility. However, if you use the input directives, you must assume increased responsibility. Input directives and options impose the following properties for programs:

- Since the DPS 7000 is a segmented memory machine, you must not place any object at a virtual address that is not in the configured memory space (the set of available segments.) *ld* invoked with only the *m* option and without an object file, gives a listing of the configured memory in the standard output file.

- *ld* will report an error if you try to mix data type sections and text type sections in the same segment.

- When the link editor is called through *cc* (1), a startup routine is linked to the user's program. This routine calls *exit* (2) after the main program is executed. If you call the link editor directly, you must ensure that the program always calls *exit* (2) rather than falling through to the end of the entry routine.

SEE ALSO

as(1), cc(1), exit(2), end(3c), a.out(4), ar(4)

"Common Object File Format" in the *Programmer's Guide*.

5. leave(1)

NAME

leave - remind you when you have to leave

SYNOPSIS

```
leave [[+] hhmm ]
```

DESCRIPTION

leave sets an alarm to a time you specify and will tell you when the time is up.

leave waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, and at the time. After the specified time, it reminds you every minute thereafter 10 more times. *leave* disappears after you log off.

You can specify the time in on of two ways, namely as an absolute time of day in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock), or you can place a + sign in front of the time, in which case the time is relative to the current time, that is, the specified number of hours and minutes from now. All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?". *leave* exits if you just type a NEWLINE, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile*.

leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use *kill -9* and its process ID.

EXAMPLES

The first example sets the alarm to an absolute time of day:

```
example% leave 1535
Alarm set for Wed Mar  7 15:35:07 1984

    ... work work work ...

example% Time to leave!
```

The second example sets the alarm for 10 minutes in the future:

```
example% leave +10 Alarm set for Wed Mar 7

    ... work work work ...

example% Time to leave!

work work work work

example% You're going to be late!
```

FILES

.login
.profile

SEE ALSO

calendar(1)

5. lex(1)

NAME

lex - lexical analysis program generator

SYNOPSIS

```
lex [ -fntv ] [ filename ] ...
```

DESCRIPTION

lex generates programs to be used in simple lexical analysis of text. Each filename (the standard input by default) contains regular expressions to search for, and actions written in C to be executed when expressions are found.

A C source program, lex.yy.c is generated, to be compiled as follows:

```
cc lex.yy.c -ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized. The actual string matched is left in *yytext*, an external character array.

Matching is done in order of the strings in the file. The strings may contain square braces to indicate character classes, as in [abx-z] to indicate a, b, x, y, and z; and the operators *, + and ?, which mean, respectively, any non-negative number, any positive number, or either zero or one occurrences of the previous character or character-class.

The dot character (.) is the class of all ASCII characters except NEWLINE.

Parentheses for grouping and vertical bar for alternation are also supported. The notation r{d,e} in a rule indicates instances of regular expression r between d and e. It has a higher precedence than |, but lower than that of *, ?, +, or concatenation. The ^ (caret character) at the beginning of an expression permits a successful match only immediately after a NEWLINE, and the $ character at the end of an expression requires a trailing NEWLINE.

The / character in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, although the remainder of the expression must follow in the input stream.

An operator character may be used as an ordinary symbol if it is within double quotes (") symbols or preceded by '\'.

Three subroutines defined as macros are expected: *input()* to read a character; *unput(c)* to replace a character read; and *output(c)* to place an output character. They are defined in terms of the standard streams, but you can override them.

The program generated is named *yylex()*, and the library contains a *main()* which calls it. The action REJECT on the right side of the rule rejects this match and executes the next suitable match; the function *yymore()* accumulates additional characters into the same *yytext*; and the function *yyless(n)* where n is the number of characters to retain in *yytext*. The macros input and output use files *yyin* and *yyout* to read from and write to, defaulted to *stdin* and *stdout*, respectively.

In a lex program, any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the *lex.yy.c* file. All rules should follow a %%, as in YACC. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be used later by surrounding it with {}.

NOTE: curly brackets do not imply parentheses; only string substitution is done.

The external names generated by *lex* all begin with the prefix yy or YY.

Certain table sizes for the resulting finite-state machine can be set in the definitions section:

%p n	number of positions is n (default 2000)
%n n	number of states is n (500)
%t n	number of parse tree nodes is n (1000)
%a n	number of transitions is n (3000)

The use of one or more of the above automatically implies the -v option, unless the -n option is used.

OPTIONS

-f Faster compilation. Do not bother to pack the resulting tables; limited to small programs.

-n Opposite of -v; -n is default.

-t Place the result on the standard output instead of in file lex.yy.c.

-v Print a one-line summary of statistics of the generated analyzer.

EXAMPLES

The following command line would draw lex instructions from the file *lexcommands*, and place the output in *lex.yy.c*:

```
lex lexcommands
```

The following:

```
%% [A-Z] putchar (yytext[0]+'a'-'A'); [ ]+$ ; [ ]+ putchar(' ');
```

is an example of a *lex* program. It converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
D       [0-9]
%%
if      printf("IF statement\n");
[a-z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

FILES

lex.yy.c

SEE ALSO

sed(1), yacc(1)

Programmer's Guide

NOTES

The *lex* command is not changed to support 8-bit symbol names, as this would produce *lex* source code that is not portable between systems.

5. line(1)

NAME

line - read one line from the standard input

SYNOPSIS

```
line
```

DESCRIPTION

line copies one line (up to a NEWLINE) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints a NEWLINE at least.

It is often used within shell scripts to read a line from the terminal.

SEE ALSO

sh(1), read(2)

5. link(1M), unlink(1M)

NAME

link, unlink - links and unlinks files and directories

SYNOPSIS

```
link file1 file2

unlink file
```

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the rm(1) and rmdir(1) commands be used instead of the *unlink* command.

The only difference between *ln(1)* and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the link(2) and unlink(2) system calls.

SEE ALSO

rm(1), link(2), unlink(2)

WARNINGS

These commands can be run only by the super-user.

5. lint(1)

NAME

lint - a C program checker

SYNOPSIS

```
lint [option] ... file ...
```

DESCRIPTION

The *lint* command attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with .c are taken to be C source files.

Arguments whose names end with .ln are taken to be the result of an earlier invocation of lint with either the -c or the -o option used. The .ln files are analogous to .o (object) files that are produced by the *cc(1P)* command when given a .c file as input. Files with other suffixes are warned about and ignored.

lint will take all the .c, .ln, and llib-lx.ln (specified by -lx) files and process them in their command line order. By default, *lint* appends the standard C lint library (llib- lc.ln) to the end of the list of files. However, if the -p option is used, the portable C lint library (llib-port.ln) is appended instead. When the -c option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the -c option is used, the .ln and the llib-lx.ln files are ignored.

Any number of lint options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

-a	Suppress complaints about assignments of long values to variables that are not long.
-b	Suppress complaints about break statements that cannot be reached. (Programs produced by lex or yacc will often result in many such complaints).
-h	Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

-lx Include additional lint library llib-lx.ln.
For example, you can include a lint version of the math library llib-lm.ln by inserting -lm on the command line. This argument does not suppress the default use of llib-lc.ln. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

-n Do not check compatibility against either the standard or the portable lint library.

-p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

-c Cause lint to produce a .ln file for every .c file on the command line. These .ln files are the product of lint's first pass only, and are not checked for inter-function compatibility.

-o lib Cause lint to create a lint library with the name llib-llib.ln. The -c option nullifies any use of the -o option. The lint library produced is the input that is given to lint's second pass.
The -o option simply causes this file to be saved in the named lint library.

To produce a llib-llib.ln without extraneous messages, use of the -x option is suggested.

The -v option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file llib-lc is written). These option settings are also available through the use of "lint comments" (see below).

The -D, -U, and -I options of *cpp(1P)* and the -g and -O options of *cc(1P)* are also recognized as separate arguments. The -g and -O options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc(1P)* command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by lint.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/	At appropriate points stops comments about unreachable code. This comment is typically placed just after calls to functions like *exit(2)*.
/*VARARGSn*/	Suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first n arguments are checked; a missing n is taken to be 0.
/*ARGSUSED*/	Turns on the -v option for the next function.
/*LINTLIBRARY*/	At the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the -v and -x options.

lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the -c option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the -c and the -o options allows for incremental use of *lint* on a set of C source files.

Generally, one invokes *lint* once for each source file with the -c option. Each of these invocations produces a .ln file which corresponds to the .c file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the -c option), listing all the .ln files with the needed -lx options. This will print all the inter-file inconsistencies. This scheme works well with *make(1P)*; it allows *make* to be used to lint only the source files that have been modified since the last time the set of source files were linted.

RETURNS

exit(2), *setjmp(3C)*, and other functions that do not return are not understood; this causes various lies.

APPLICATION USAGE

If the TOOLS_PATH environment variable is set at a directory path, lint will search the called tools under the specified directory. (See the Cross Program Production chapter in the Programmer's Guide).

FILES

/usr/lib/lint[12]	First and second passes
/usr/lib/llib-lc.ln	Declarations for C Library functions (binary format; source is in /usr/lib/llib-lc)
/usr/lib/llib-port.ln	Declarations for portable functions (binary format; source is in /usr/lib/llib-port)
/usr/lib/llib-lm.ln	Declarations for Math Library functions (binary format; source is in /usr/lib/llib-lm)
/usr/tmp/*lint*	Temporaries. The temporary directory can be specified by setting the environment variable TMPDIR (see tempnam(3S)).

SEE ALSO

cc(1), cpp(1), make(1P).

5. ln(1)

NAME

ln - make links

SYNOPSIS

```
ln sourcename [targetname]
ln sourcename1 sourcename2 [sourcename3 ...] targetdirectory
```

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There are two kinds of links; hard links and symbolic links.

By default *ln* makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

WARNING

The -s option (to create symbolic links) is NOT SUPPORTED BY OPEN 7.

Given one or two arguments, *ln* creates a link to an existing *sourcename*. If *targetname* is given, the link has that name; *targetname* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *sourcename*.

Given more than two arguments, *ln* makes links in *targetdirectory* to all the named source files. The links made will have the same name as the file being linked to.

SEE ALSO

cp(1), mv(1), rm(1).

5. loadatoe(1), loadetoa(1)

NAME

loadatoe, loadetoa - load the conversion table information for ASCII-to-EBCDIC and EBCDIC-to-ASCII data transfers.

SYNOPSIS

```
loadatoe <conversion table file>

loadetoa <conversion table file>
```

DESCRIPTION

OPEN 7 uses a character conversion table to ensure functional interoperability between GCOS 7 and OPEN 7. This table determines the character equivalents between EBCDIC characters and the ASCII character set. All data exchanges between GCOS 7 and OPEN 7 (CNDSA, FORMC, XFORM, FTP, etc.) use this table.

The *loadatoe* and *loadetoa* commands are provided for the purpose of generating a new character conversion table:

- **loadatoe** loads the conversion table information for ASCII-to-EBCDIC data transfers.
- **loadetoa** loads the conversion table information for EBCDIC-to-ASCII data transfers.

These commands provide you with a means of adapting the character conversion table to the particular requirements of your site. The resultant table information is loaded into the OPEN 7 kernel to replace the default conversion table.

NOTE: The new table information will be lost each time OPEN 7 is re-booted.

The *loadetoa* and *loadatoe* commands can be entered at any time from within an OPEN 7 session. These commands require superuser privilege.

MESSAGES

Result Messages:

Both **loadetoa** and **loadatoe** verify the existence of the conversion file, user access privileges and the integrity of the new conversion table file (which contains the 256 hexadecimal values).

These commands first verify the existence of the file **/etc/etoa.loc** or **/etc/atoe.loc**, and that the addresses contained in the **/etc/etoa.loc** and **atoe.loc** files are correct. If not, the procedure is aborted.

Upon completion **loadetoa** displays the message twice:

```
loadetoa: Writing new table at Address <kernel address> successful
loadetoa: Old table saved in /etc/etoa.sav
```

The command **loadetoa** creates two files or updates: **/etc/etoa.loc** contains the kernel address of the conversion table and **/etc/etoa.sav** contains the contents of the previous conversion table.

Upon completion, **loadatoe** displays the message:

```
loadatoe: Writing new table at Address <kernel address> successful
loadatoe: Old table saved in /etc/etoe.sav
```

The command **loadatoe** creates two files: **/etc/atoe.loc** contains the kernel address of the conversion table and **/etc/atoe.sav** contains the contents of the previous conversion table.

Error messages:

Both **loadetoa** and **loadatoe** verify the existence of the conversion file, user access privileges and the integrity of the new conversion table file (which contains the 256 hexadecimal values).

These commands first verify the existence of the file **/etc/etoa.loc** or **/etc/atoe.loc**, and that the addresses contained in the **/etc/etoa.loc** and **atoe.loc** files are correct. If not, the procedure is aborted.

FILES

Creating a New Character Conversion Table

Before you can replace OPEN 7's default character conversion table you must first create a file containing the new character values.

The new table information should be created as text file containing the hexadecimal equivalents for each ebcdic character. The following figure illustrates the contents of this type of file.

NOTE: This file must only contain the 256 hexadecimal character values.

```
ed
a
00 01 02 03 4c 09 86 7f 97 8d 8e 0b 0c 0d 0e 0f
10 11 12 13 9d 85 08 87 18 19 92 8f 1c 1d 1e 1f
80 81 82 83 84 0a 17 1b 88 89 8a 8b 8c 05 06 07
90 91 16 93 94 95 96 04 98 99 9a 9b 14 15 9e 1a
20 a0 a1 a2 a3 a4 a5 a6 a7 a8 5b 2e 3c 28 2b 21
26 a9 aa ab ac ad ae af b0 b1 5d 24 2a 29 3b 5e
2d 2f b2 b3 b4 b5 b6 b7 b8 b9 7c 2c 25 5f 3e 3f
ba bb bc bd be bf c0 c1 c2 60 3a 23 40 27 3d 22
c3 61 62 63 64 65 66 67 68 69 c4 c5 c6 c7 c8 c9
ca 6a 6b 6c 6d 6e 6f 70 71 72 cb cc cd ce cf d0
d1 7e 73 74 75 76 77 78 79 7a d2 d3 d4 d5 d6 d7
d8 d9 da db dc dd de df e0 e1 e2 e3 e4 e5 e6 ef
7b 41 42 43 44 45 46 47 48 49 e8 e9 ea eb ec ed
7d 4a 4b 4c 4d 4e 4f 50 51 52 ee ef f0 f1 f2 f3
5c 9f 53 54 55 56 57 58 59 5a f4 f5 f6 f7 fa f9
30 31 32 33 34 35 36 37 38 39 fa fb fc fd fe ff
.
w etoa.sample
q
```

SEE ALSO

atoe(3C), etoa(3C)

5. login(1)

NAME

login - sign on

SYNOPSIS

```
login [ name [ env-var ... ]]
```

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a ctrl-d to indicate an "end-of-file".

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing the following command from the initial shell:

```
exec login
```

login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */etc/profile* is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh(1)*) is initialized, and the file .profile in the working directory is executed, if it exists. These specifications are found in the */etc/passwd* file entry for the user.

The name of the command interpreter is - followed by the last component of the interpreter's path name (i.e., -sh). If this field in the password file is empty, then the default command interpreter, */bin/sh* is used. If this field is "*", then the named directory becomes the root directory, the starting point for path searches for path names beginning with a /. At that point login is re-executed at the new level which must have its own root structure, including */etc/login* and */etc/passwd*.

The basic environment is initialized to:

HOME=your-login-directory
PATH=:/bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification
TERM=terminal type

The environment may be expanded or modified by supplying additional arguments to login, either at execution time or when login requests your login name. The arguments may take either the form xxx or xxx=yyy. Arguments without an equal sign are placed in the environment as

Ln=xxx

where n is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables PATH and SHELL cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/spool/mail/your-name	mailbox for user your-name
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile

NOTES

C2 Secure Environment

In a Secure environment, this command can be used only by the user with authentication subsystem privilege.

The *login* command has been moved to the /tcb/lib directory.

SEE ALSO

csh(1), ksh(1), mail(1), sh(1), su(1M), passwd(4)
profile(4) and environ(5) in the *Programmer's Reference Manual*.

DIAGNOSTICS

login incorrect
If the user name or the password cannot be matched.

No shell, cannot open password file, or no directory
Consult a Bull system programming counselor.

No utmp entry
You must exec "login" from the lowest level "sh" if you attempted to execute login as a command without using the shell's exec internal command or from other than the initial shell.

5. logname(1)

NAME

logname - get login name

SYNOPSIS

```
logname
```

DESCRIPTION

logname writes the user's login name on the standard output. It corresponds to the $LOGNAME variable, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).

logname(3X), environ(5) in the *Programmer's Reference Manual*.

5. lp(1)

NAME

lp - sends a file to the printer spooler to print OPEN 7 reports on OPEN 7 printers or the GCOS 7 SYSOUT.

cancel - cancels print requests

SYNOPSIS

OPEN 7 printers:

```
lp -d dest [-c] [-m] [-s] [-w] [-n number] [-t title]
           [-o option] files
```

GCOS 7 printers (SYSOUT):

```
lp -d SYSOUT [-c] [-m] [-s] [-w] [-n number] [-t title]
             [-C output_class] [-X banner_line2]
             [-Y banner_line3] [-Z banner_line4] [-S print_site]
             [-T print_station] [-P priority] [-N name]
             [-D printer_type ] files
```

Cancel print request:

```
cancel [request_ids] [printers]
```

DESCRIPTION

lp arranges for the named files and associated information (collectively called a request) to be printed by an OPEN 7 printer (first form of the command), or by a GCOS 7 printer (second form of the command). If no file names are mentioned, the standard input file is assumed. The file name "-" stands for the standard input file and may be supplied on the command line in conjunction with named files. Files are printed in the order in which they appear.

lp associates a unique identity with each request and prints it in the standard output file. This identity can be used later to cancel (see *cancel*) or find the status (see *lpstat* (1)) of the request (see Application notes below).

The options and arguments are:

-d dest	Choose *dest* as the printer or class of printer that is to do the printing. Under certain conditions (printer not available, file space insufficient, and so on), requests for specific destinations may not be accepted (see *accept* (1M) and *lpstat* (1M)). By default, *dest* is taken from the environment variable LPDEST if it is set. Otherwise, a default destination (if one exists) for the OPEN 7 system is used. Acceptable destination names are: LP: this stands for the class containing all OPEN 7 printers (see Application Notes below). The request will be printed on the first available OPEN 7 printer ; *lpi* (where i is a digit): This stands for OPEN 7 printer *lpi* and allows the user to choose a specific printer; use *lpstat* (1) to find out which printers are available on your system; SYSOUT: this stands for GCOS 7 sysout
-c	Make copies of the files immediately *lp* is invoked. Normally, files are not copied, but are linked whenever possible. If the *-c* option is not given, then you should be careful not to remove any of the files before the request has been fully printed. It should also be noted that in the absence of the *-c* option, any changes made to the named files after the request is made but before printing will be reflected in the printed output.
-m	Send mail (see *mail* (1)) after the files have been printed (see Application notes below). By default no mail is sent upon normal completion of the print request.
-w	Write a message on the user's terminal after the files have been printed (see Application notes below). If the user is not logged in, then mail will be sent instead.
-s	Suppress messages from *lp* such as *request id is ...*.
-n number	Print *number* of copies (default is 1) of the output.
-t title	Print *title* on the banner page of the output. If *dest* is SYSOUT, *title* is used as the first line of the output banner and will be truncated to 12 characters.
-o option	Specifies printer-dependent options. Several such options may be collected by specifying the *-o* keyletter more than once. For more information about what is valid for options, see Models in *lpadmin* (1M). The *-o* keyletter is not allowed when *dest* is SYSOUT.

The following options are available only when *dest* is SYSOUT (printing via the GCOS 7 SYSOUT). Unless explicitly specified, their default values are those set in the GCOS 7 system configuration and in the JCL used to start OPEN 7.

-C output_class	Specifies the GCOS 7 output class to be used (letter A through Z).
-X banner_line2	specifies the string (up to 12 characters, extra characters will be truncated) to be used as the second line of the output banner.
-Y banner_line3	Specifies the string (up to 12 characters, extra characters will be truncated) to be used as the third line of the output banner. If this option is absent the *lp* command uses the USER environment variable if it exists, otherwise the login name is used.
-Z banner_line4	Specifies the string (up to 12 characters, extra characters will be truncated) to be used as the fourth line of the output banner.
-S print_site	Specifies the GCOS 7 system (up to 8 characters) where the output is to be redirected.
-T print_station	Specifies the station (up to 8 characters) where the output is to be redirected.
-P priority	Specifies the selection priority for GCOS 7 output; may be 1 to 9.
cancel	Cancel print requests (see Application Notes below) that were made by the *lp* command. The command line arguments may be either *request_ids* (as returned by *lp*) or printer names (for a complete list, see *lpstat* (1)). Specifying a *request_id* cancels the associated request even if it is currently printing. Specifying a printer (*lp1*, for example) cancels the request that is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.
-N name	When the destination is SYSOUT, this specifies the name of GCOS 7 output that will be writen on the GCOS 7 side when the command DO (Display_output) is used for the OPEN 7 job.
-D printer_type	When the destination is SYSOUT, this specifies the GCOS printer_type in GCOS 7 syntax. This option is useful for indicating the number of columns to be used. Example: -D PR/H160 allows printing on 160 columns. Default value is 132 columns.

APPLICATION NOTES

Without the -N option, the GCOS 7 output appears with the OPEN 7 user name when the GCOS 7 command DO (Display_output) is used for the OPEN 7 job.

Requests submitted with the destination SYSOUT may be followed by the LP spooling system until they have been written to the GCOS 7 SYSOUT. This means:

- if the *-m* option or the *-w* option is given, the mail or the message is sent after the files have been written in the GCOS 7 SYSOUT and not after they have actually been printed by GCOS 7.

- the *cancel* and *lpstat*(1) commands may be used for SYSOUT requests only until they have been written in the GCOS 7 SYSOUT.

On your system, OPEN 7 printers may be subdivided into more than one class if there are different types of printer. Use *lpstat*(1) to find out what classes are available.

FILES

`/usr/spool/lp/*`

EXAMPLES

Printing on an OPEN 7 printer:

```
lp -d LP -t "program" program.c
```

Printing on a GCOS 7 printer:

```
lp -d SYSOUT -t "program" program.c
```

SEE ALSO

enable(1), lpstat(1), mail(1), accept(1M), lpadmin(1M), lpsched(1M), lp(7), lpsetup(1M)

5. lp(7)

NAME

lp - the driver for accessing GCOS 7 SYSOUT printers

DESCRIPTION

The OPEN 7 standard UNIX spooler can send its listing to the GCOS 7 SYSOUT printer using the SYSOUT driver (see *lp*(1)).

The fileset */dev/lp** (major device 3) provides the interface with the GCOS 7 SYSOUT.

The driver interprets only linefeed and formfeed characters. The default line length is 256 columns per line; lines longer than 256 characters are continued on next line. Remember that the GCOS 7 SYSOUT has its own line length rules, depending on the type of printer (132, 136, 160 columns). Lines longer than the device capabilities are wrapped round.

Two *ioctl*(2) system calls are available:

```
#include <sys/prout.h>
ioctl (fildes, command, &prsysout);
struct prsysout prsysout;
```

The commands available are:

SCGETA	Use the GCOS 7 default values. The values are loaded in the fields of structure *prsysout*. The default parameters are those set in the GCOS 7 system configuration (OWCLASS, OWDEVICE, and so on) and modified by the command OVL in the OPEN 7 boot JCL.
SCSETA	Modify the default values for the SYSOUT output. This command must be issued between the OPEN and the first PUT.

prsysout structure

The purpose of this structure is to inform an OPEN 7 user of the default output parameters (SCGETA) or to permit overriding (SCSETA). For a full explanation of these parameters see the *JCL Reference Manual* (SYSOUT command).

```
#define SIOC ('S'<<8)
#define SCGETA (SIOC|1)
#define SCSETA (SIOC|2)

struct prsysout
{
 char class; /* output class of output: letters A through Z */
```

```
#define PR_DFLT_PRIORITY 0xff
 char priority; /* output priority values 0 through 7 /*
                /* (highest is 0) */

 char name[8]; /* symbolic name associated with the output file */
#define PR_BANNER 0x01 /* print banner pages */
#define PR_NBANNER 0x00 /* no print of banner pages */
 char banner; /* print banner pages */

 char btext1[12]; /* text of first line banner */
 char btext2[12]; /* text of second line banner */
 char btext3[12]; /* text of third line banner */
 char btext4[12]; /* text of forth line banner */

#define PR_HOLD 0x01  /* hold output */
#define PR_NHOLD 0x00 /* release output depending on parameter */
                      /* nwhen */
 char hold; /* hold output pending GCOS 7 RO command */

#define PR_IMMED 0x04
#define PR_EARLY 0x14
 char nwhen; /* point of notification of sysout */
             /* IMMED - notification after closed */
             /* EARLY - notification after EARLYNB pages */

 char rfu[3]; /* for alignment purposes */

 int earlynb; /* nb pages for each notification */
              /* must be 0 for all points of notification */
              /* other than EARLY */
              /* and significant( >0 ) for EARLY */

#define PR_CD   "CD/P/C80"
#define PR_H160 "PR/H160"
#define PR_H132 "PR/H132"
#define PR_H120 "PR/H120"
#define PR_PR   "PR"
 char device[40]; /* printer device class and attributes */

 char copies; /* number of copies to be produced */
              /* max value: 99 */
 char media[6]; /* belt character set and paper identification */

 char hostname[8]; /* name of the host system */

 char station[8]; /* name of the logical station (RBF) */

#define PR_NSLEW 0x01
#define PR_SLEW 0x00
 char nslew; /* every skip function transformed to new line */

};
```

Correspondence between:

prsysout structure	*JCL command SYSOUT*
class	CLASS
priority	PRIORITY
name	NAME
banner	BANNER, NBANNER
btext1	BANINF
btext2	BANINF
btext3	BANINF
btext4	BANINF
hold	HOLD, NHOLD
nwhen	WHEN=IMMED
earlynb	WHEN=digit5
device	DEVCLASS
copies	COPIES
media	MEDIA
hostname	DEST
station	STATION

CONFIGURATION

major device: 3

The number of minor entries is given by *define NOUT* in *sys/config.h*. The default value is 8 (#define NPT 8).

The minor device ranges from 0 to 31, with 0 through 7 as configuration value.

FILES

```
/dev/lp*
/usr/include/sys/prout.h
```

SEE ALSO

lp(1)

5. lpadmin(1M)

NAME

lpadmin - configure the LP spooling system

SYNOPSIS

```
/usr/lib/lpadmin -p printer [options]
/usr/lib/lpadmin -x dest
/usr/lib/lpadmin -d[dest]
```

DESCRIPTION

lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination.

lpadmin must not be used when the LP scheduler *lpsched(1M)* is running, except where noted below.

Exactly one of the -p, -d or -x options must be present for every legal invocation of *lpadmin*.

-d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, there is no system default destination. This option may be used when *lpsched(1M)* is running. No other options are allowed with **-d**.

-x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other options are allowed with **-x**.

-p*printer* names a printer to which all of the options below refer. If *printer* does not exist, then it will be created.

The following options are only useful with **-p** and may appear in any order. For ease of discussion, the printer will be referred to as P below.

-c*class* inserts printer P into the specified *class*. *class* will be created if it does not already exists.

-e*printer* copies an existing printer's interface program to be the new interface program for P.

-h indicates that the device associated with P is hardwired. this option is assumed when creating a new printer, unless the **-l** option is supplied.

-i*interface* establishes a new interface program for P. *interface* is the pathname of the new program.

-l indicates that the device associated with P is a login terminal. The LP scheduler *lpsched(1M)*, disables all login terminals automatically each time it is started. Before re-enabling P, its current device should be established using *lpadmin*.

-m*model* selects a model interface program for P. *model* is one of the model interface names supplied with the LP software (see **Models** below).

-r*class* removes printer P from the specified *class*. If P is the last member of the class, then the class will be removed.

-v*device* associates a new device with printer P. *device* is the pathname of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from associating the same device with more than one printer. If only the **-p** and **-v** options are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions

When creating a new printer, the **-v** option and one of the **-e**, **-i** or **-m** options must be supplied. Only one of the **-e**, **-i** or **-m** options must be supplied. The **-h** and **-l** options are mutually exclusive. Printer and class names must be no longer than 14 characters and must consist entirely of the characters A-Z, a-z, 0-9 and _ (underscore).

Models

Model printer interface programs are supplied with the LP software. They are shell procedures which interface between *lpsched(1M)* and devices. All models reside in the directory */usr/spool/lp/model* and may be used as is with *lpadmin -m*. Models should have 644 permission if owned by lp & bin, or 664 permission if owned by bin & bin. Alternatively, LP administrators may modify copies of models and then use *lpadmin -i* to associate them with printers.

To see the list of supported printers, refer to the *printers(8)* command.

EXAMPLES

```
/usr/lib/lpadmin -ppr1 -m1232                        (1)

/usr/lib/lpadmin -pst1 -v/dev/ttyc3d6 -m1260s        (2)

nroff -T450 files | lp -dst1 -of              }
nroff -T450-12 files | lp -dst1 -of           }      (3)
nroff -T37 files | col | lp -dst1             }
```

(1) Assuming there is an existing Bull PRT1232 line printer named pr1, it will use the 1232 model interface after this command.

(2) Use this command to add a NIP3 PRT1260, serial interface printer called st1 to the LP configuration.

(3) Use any of these commands to print an nroff document on st1.

FILES

/usr/spools/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1), nroff(1), printers(8).

5. lpc(1M)

NAME

lpc - line printer control program

SYNOPSIS

```
/etc/lpc [ command [ parameter...  ] ]
```

DESCRIPTION

lpc controls the operation of the printer, or of multiple printers, as described in the /etc/printcap database. lpc commands can be used to start or stop a printer, disable or enable a printer's spooling queue, rearrange the order of jobs in a queue, or display the status of each printer - along with its spooling queue and printer daemon.

With no arguments, lpc runs interactively, prompting with " lpc> ". If arguments are supplied, lpc interprets the first as a command to execute; each subsequent argument is taken as a parameter for that command. The standard input can be redirected so that lpc reads commands from a file.

USAGE

Commands may be abbreviated to an unambiguous substring.

all | [printer ...] means that the command applies to all printers, or to the printers whose name is given.

NOTE: the printer parameter is specified just by the name of the printer (like lw), not as you would specify it to lpr(1) or lpq(1) (i.e. not as -Plw).

? [command]...	help [command]...Display 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(1)) for the specified printers. The abort command can only be used by a super-user.
clean [all | [printer...]]	Remove all files with names beginning with cf, tf, or df from the specified printer queue(s) on the local machine. The clean command can only be used by a super-user.
disable [all | [printer...]]	Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by lpr(1). The disable command can only be used by a super-user.

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(1V). This is normally used to take a printer down and let others know why (lpq(1) indicates that the printer is down, as does the status command).

enable [all | [printer...]]
: Enable spooling on the local queue for the listed printers, so that lpr(1) can put new jobs in the spool queue. The enable command can only be used by a super-user.

exit
: Exit from lpc.

quit
: Exit from lpc.

restart [all | [printer...]]
: Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. lpq(1) reports that there is no daemon present when this condition occurs. This command can be run by any user.

start [all | [printer...]]
: Enable printing and start a spooling daemon for the listed printers. The start command can only be used by a super-user.

status [all | [printer...]]
: Display the status of daemons and queues on the local machine. This command can be run by any user.

stop [all | [printer...]]
: Stop a spooling daemon after the current job completes and disable printing. The stop command can only be used by a super-user.

topq printer [job#...] [user...]
: Move the print job(s) specified by job# or those job(s) belonging to user to the top (head) of the printer queue. The topq command can only be used by a super-user.

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

SEE ALSO

lpq(1), lpr(1), lprm(1), printcap(5), lpd(1M)

DIAGNOSTICS

?Ambiguous command
The abbreviation you typed matches more than one command.

?Invalid command
You typed a command or abbreviation that was not recognized.

?Privileged command
You used a command which can be executed only by a super-user.

5. lpclean(1M)

NAME

lpclean - cleans and reconfigures the LP spooling system

SYNOPSIS

```
/usr/lib/lpclean [-s]
```

DESCRIPTION

The *lpclean* procedure is called to reconfigure the LP spooling system.

If the ***-s*** option is given, printing requests pending are saved and resubmitted after LP reconfiguration.

If no OPEN 7 printers have been installed using the *lpsetup*(1M) command, *lpclean* resets only printers of SYSOUT class, which allow printing of OPEN 7 reports on GCOS 7 system printers.

If OPEN 7 printers have been installed using *lpsetup*(1M), they are also reset by *lpclean*.

FILES

```
/usr/spool/lp/*
/tmp/spool/*
/etc/config.lp
```

RESTRICTIONS

Requests which had no title originally are printed with the LP administrator's login name upon resubmission. In addition, if mail was asked for in the original request (*lp*(1) options *m* or *w*) it will be sent to the LP administrator after resubmission.

SEE ALSO

initlp(1M), lpsetup(1M), lpsched(1M), lpshut(1M)

5. lpd(1M)

NAME

lpd - printer daemon

SYNOPSIS

```
/usr/lib/lpd [-l] [-L logfile] [port#]
```

DESCRIPTION

lpd is the line printer daemon (spool area handler). It is usually invoked at boot time from the *rc(8)* script, making a single pass through the *printcap(5)* file to find out about the existing printers and printing any files left after a crash. It then accepts requests to print files in a queue, transfer files to a spooling area, display a queue's status, or remove jobs from a queue. In each case, it forks a child process for each request, and continues to listen for subsequent requests.

The Internet port number used to communicate with other processes is usually obtained with *getservent(3N)*, but can be specified with the *port#* argument.

If a file cannot be opened, an error message is logged using the LOG_LPR facility of *syslog(3)*. *lpd* tries up to 20 times to reopen a file that it expects to be present, after which it proceeds to the next file or job.

OPTIONS

-l	Log valid requests received from the network. This can be useful for debugging purposes.
-L logfile	Change the file used for writing error conditions to logfile. The default is to report a message using the *syslog(3)* facility.
port#	may be used to specify the Internet port number used to communicate with other processes. But usually this port number is obtained with *getservent(3N).*

OPERATION

Access Control

Access control is provided by two means. First, all requests must come from one of the machines listed in either the file */etc/hosts.equiv* or */etc/hosts.lpd*. (The latter file is in *hosts.equiv(5)* format.) Second, if the *rs* capability is specified in the printcap entry, *lpr(1)* requests are only honored for users with accounts on the printer host.

Lock File

The lock file in each spool directory is used to prevent multiple daemons from becoming active, and to store information about the daemon process for *lpr(1)*, *lpq(1)*, and *lprm(1)*.

lpd uses *flock(2)* 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 file name of the current job being printed. The second line is updated to reflect the current status of lpd for the programs *lpq(1)* and *lprm(1)*.

Control Files

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 that indicates what to do with the remainder of the line:

J Job name to print on the burst page.

C Classification line on the burst page.

L Literal. This line contains identification information from the password file, and causes a burst page to be printed.

T Title string for page headings printed by *pr(1)*.

H Hostname of the machine where *lpr(1)* was invoked.

P Person. Login name of the person who invoked *lpr(1)*. This is used to verify ownership by *lprm(1)*.

M Send mail to the specified user when the current print job completes.

f Formatted File, the name of a file to print that is already formatted.

l Like f, but passes control characters along, and does not make page breaks.

p Name of a file to print using *pr(1)* as a filter.

W Width. Changes the page width (in characters) used by *pr(1)* and the text filters.

I Indent. Specify the number of characters by which to indent the output

U Unlink. The name of file to remove upon completion of printing.

N Filename. The name of the file being printed, or a blank for the standard input (when *lpr(1)* is invoked in a pipeline).

Data Files

When a file is spooled for printing, the contents are copied into a data file in the spool directory. Data file names begin with df. When lpr is called with the -s option, the control files contain a symbolic link to the actual file, and no data files are created.

Minfree File
The file minfree in each spool directory contains the number of kilobytes to leave free so that the line printer queue won't completely fill the disk.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/min	free minimum free space to leave
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	hosts allowed equivalent host access
/etc/hosts.lpd	hosts allowed printer access only

SEE ALSO

lpq(1), lpr(1), lprm(1), hosts(4), hosts.equiv(4), printcap(5), lpc(1M), pac(1M)

5. lpdinstall(1M)

Installs or uninstalls a printer given by its name. Referred to by lpstart(1M) and lpstop(1M).

5. lpinstall(1M)

/usr/lib/lpinstall is a shell procedure which configures a printer in the LP spooling system. See lpstart(1M) and lpstop(1M).

5. lplist(1M)

NAME

lplist - gives the status of existing line printers

SYNOPSIS

```
/usr/lib/lplist
```

DESCRIPTION

The *lplist* command reports the status of the line printers existing in the configuration with two formats:

1. the status of installed printers, which is the status reported by the *lpstat(1)* command,

2. the status of configured printers, which is the configuration of each printer found in the */etc/config.lp* file.

EXAMPLES

1. Assuming there is an existing Bull PRT1232 line printer named pr1, the result is the following:

```
    installed printers :
printer pr1 is idle.   enabled since feb 20 10:50
   configured printers :
pr1  on  class=c11, line=pr1-ttyc3d7, model=m-1232,
            term= -   : STARTED
```

2. Assuming there is an existing Bull PRT1232 line printer named pr1, and an existing NIP2 PRT1253 line printer named nip, the result is the following:

```
    installed printers :
printer pr1 is idle.   enabled since feb 20 10:50
printer nip disabled since feb 23 15:17
   configured printers :
pr1  on   class=c11, line=pr1-ttyc3d7, model=m-1232,
             term= -   : STARTED
nip  off  class=c12, line=pr2-ttyc3d8, model=m-1253s,
```

FILES

/etc/config.lp

SEE ALSO

lpstart(1M), lpstop(1M), lpstat(1)

5. lpmove(1M), lpsched(1M), lpshut(1M)

NAME

lpmove - move LP requests
lpsched, lpshut - start/stop the LP request scheduler

SYNOPSIS

```
/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2
```

DESCRIPTION

lpsched schedules requests taken by *lp(1)* for printing on line printers.

lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

lpmove moves requests that were queued by *lp(1)* between LP destinations. This command must be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination *dest*. *Requests* are requests ids as returned by *lp(1)*. The second form moves all requests from destination *dest1* to destination *dest2*. As a side effect, *lp(1)* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept(1M)*) for the new destination when moving requests.

FILES

/usr/spool/lp/*

NOTES

C2 Secure Environment
In a Secure environment, *lpmove* can be used only by the superuser with *line printer* subsystem privilege.

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1)

5. lpq(1)

NAME

lpq - display the queue of printer jobs

SYNOPSIS

```
lpq [-Pprinter] [-l] [+ [interval]] [job# ... ]  [username... ]
```

DESCRIPTION

lpq displays the contents of a printer queue. It reports the status of jobs specified by job#, or all jobs owned by the user specified by username. *lpq* reports on all jobs in the default printer queue when invoked with no arguments.

For each print job in the queue, *lpq* reports the user's name, current position, the names of input files comprising the job, the job number (by which it is referred to when using *lprm(1)*) and the total size in bytes. Normally, only as much information as will fit on one line is displayed. Jobs are normally queued on a first-in-first-out basis. Filenames comprising a job may be unavailable, such as when lpr is used at the end of a pipeline; in such cases the filename field indicates "(standard input)".

If *lpq* warns that there is no daemon present (i.e. due to a malfunction), the *lpc(8)* command can be used to restart a printer daemon.

OPTIONS

-P**printer**	Display information about the queue for the specified **printer**. In the absence of the -P option, the queue to the printer specified by the PRINTER variable in the environment is used. If the PRINTER variable isn't set, the queue for the default printer is used.
-l	Display queue information in long format; includes the name of the host from which the job originated.
+[interval]	Display the spool queue periodically until it empties. This option clears the terminal screen before reporting on the queue. If an interval is supplied, *lpq* sleeps that number of seconds in between reports.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/l*	spooling directory, as determined from printcap
/usr/spool/l*/cf*	control files specifying jobs
/usr/spool/l*/lock	lock file to obtain the currently active job

SEE ALSO

lpr(1), lprm(1), lpc(1M), lpd(1M)

DIAGNOSTICS

printer is ready and printing
The *lpq* program checks to see if there is a printer daemon. If the daemon is hung, the super-user can abort the current daemon and start a new one using *lpc(1M)*.

Waiting for printer to become ready.
The daemon could not open the printer device. The printer may be turned off-line. This message can also occur if a printer is out of paper, the paper is jammed, etc. Another possible cause is that a process, such as an output filter, has exclusive use of the device. The only recourse in this case is to kill the offending process and restart the printer with lpc.

waiting for host to come up
A daemon is trying to connect to the remote machine named host, in order to send the files in the local queue. If the remote machine is up, lpd on the remote machine is probably dead or hung and should be restarted using lpc.

sending to host
The files are being transferred to the remote hosts, or else the local daemon has hung while trying to transfer the files.

Warning: printer is down
The printer has been marked as being unavailable with lpc.

Warning: no daemon present
The lpd process overseeing the spooling queue, as indicated in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. Check the printer's error log for a diagnostic from the deceased process; you can restart the printer daemon with lpc.

BUGS

lpq may report unreliably. The status as reported may not always reflect the actual state of the printer. Under some circumstances, *lpq* reports that a printer is ready and printing when the daemon is, in fact, hung.

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

lpq is sometimes unable to open various files when the lock file is malformed.

5. lpr(1)

NAME

lpr - send a job to the printer

SYNOPSIS

```
lpr [-Pprinter] [-#_copies] [-C_class] [-J_job] [-T_title]
    [-i [_ident]] [-w_cols] [-r] [-m] [-h]
    [-_filter-option] [_filename ... ]
```

DESCRIPTION

The *lpr* command creates a printer job in a spooling area for subsequent printing as facilities become available. Each printer job consists of a control file and one or more data files.

The data files are copies of (or, with -s , symbolic links to) each **filename** you specify. The spool area is managed by the line printer daemon, *lpd(1M)*. Jobs that specify a printer on a remote machine are forwarded by *lpd*. *lpr* reads from the standard input if no files are specified.

OPTIONS

-P**printer** — Send output to the named **printer**. Otherwise, send output to the printer named in the PRINTER environment variable, or to the default printer, lp.

-#**copies** — Produce the number of **copies** indicated for each named file. For example:

```
lpr -#3 index.c lookup.c
```

produces three copies of index.c, followed by three copies of lookup.c. On the other hand, the following command:

```
cat index.c lookup.c | lpr -#3
```

generates three copies of the concatenation of the files.

-C**class** — Print **class** as the job classification on the burst page. For example:

```
lpr -C Operations new.index.c
```

replaces the system name (the name returned by hostname) with "Operations" on the burst page, and prints the file `new.index.c` .

-J**job** — Print job as the **job** name on the burst page. Normally, lpr uses the first file's name.

-T**title** Use **title** instead of the file name for the title used by pr(1V).

-i[**ident**] Indent output by **ident** SPACE characters. Eight SPACE characters is the default. The indent is passed to the input filter. If no input filter is present, this option is ignored.

-w**cols** Use **cols** as the page width for pr.

-r Remove the file upon completion of spooling or upon completion of printing with the -s option.

-m Send mail upon completion.

-h Suppress printing the burst page.

filter-option The following single letter options 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.

-p Use pr to format the files (lpr -p is very much like `pr | lpr').
-l Print control characters and suppress page breaks.

If no **filter-option** is given (and the printer can interpret PostScript), the string `%!' as the first two characters of a file indicates that it contains PostScript commands.

These filter options offer a standard user interface, and all options may not be avaiable for, nor applicable to, all printers.

NOTE: The -s option is not supported by OPEN 7.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities data base
/usr/lib/lpd	line printer daemon
/var/spool/l*	directories used for spooling
/var/spool/l*/cf*	daemon control files
/var/spool/l*/df*	data files specified in `cf' files
/var/spool/l*/tf*	temporary copies of `cf' files

SEE ALSO

lpq(1), lprm(1), pr(1), printcap(5), lpc(1M), lpd(1M)

DIAGNOSTICS

lpr: copy file is too large
A file is determined to be too "large" to print by copying into the spool area. lpr truncates the file, and prints as much of it as it can. The maximum file length is specified by the mx capability of the printcap(5) entry for the printer. If no mx capability is specified, the default limit is 1000 Kbytes. Use the -s option as defined above to make a symbolic link to the file instead of copying it.

lpr: **printer**: unknown printer
The printer was not found in the printcap database. Usually this is a typing mistake. However, it may indicate a missing or incorrect entry in the `/etc/printcap` file.

lpr: **printer:** jobs queued, but cannot start daemon
The connection to lpd on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket `/dev/printer` to be sure it still exists (if it does not exist, there is no lpd process running).

lpr: **printer** printer queue is disabled
This means the queue was turned off with `/etc/lpc disable` **`printer,`** to prevent lpr from putting files in the queue.
This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with *lpc*. If a connection to *lpd* on the local machine cannot be made, lpr responds that the daemon cannot be started. Diagnostics may be printed in the daemon log file regarding missing spool files by *lpd*.

BUGS

Command line options cannot be combined into a single argument as with some other commands. The two commands below are not equivalent:

```
lpr -fs
lpr -f -s
```

Placing the -s flag first, or writing each option as a separate argument, makes a link as expected.

lpr -p is not precisely equivalent to pr | lpr. lpr -p puts the current date at the top of each page, rather than the date last modified.

Fonts for *troff(1)* and T X(R) reside on the printer host. It is not possible to use local font libraries.

lpr refuses to print a.out files and library archives.

The -s option only avoids copying the data file to the spool directory of the local machine. If the printer for a job resides on a remote machine, the data file will be copied to the remote spool directory in all cases.

5. lprm(1)

NAME

lprm - remove jobs from the printer queue

SYNOPSIS

```
lprm [ -Pprinter ] [ - ] [ job# ...  ] [ username ...  ]
```

DESCRIPTION

The *lprm* command removes a job or jobs from a printer's spooling queue. Since the spool directory is protected from users, using *lprm* is normally the only method by which a user can remove a job.

With no arguments, *lprm* deletes the job that is currently active, provided that the user who invoked *lprm* owns that job.

When the super-user specifies a username, *lprm* removes all jobs belonging to that user.

You can remove a specific job by supplying its job number as an argument, which you can obtain using *lpq(1)*. For example:

```
example% lpq -Pprinter
host is ready and printing
Rank    Owner  Job  Files                    Total Size
active  wendy  385  standard input           35501 bytes
example% lprm -Pprinter 385
```

lprm reports the names of any files it removes, and is silent if there are no applicable jobs to remove. *lprm* kills the active printer daemon, if necessary, before removing spooled jobs; it restarts the daemon when through.

OPTIONS

-P *printer* — Specify the queue associated with a specific **printer**. Otherwise the value of the PRINTER variable in the environment is used. If this variable is not set, the queue for the default printer is used.

- — Remove all jobs owned by you. If invoked by a super-user, all jobs in the spool are removed. (Job ownership is determined by the user's login name and host name on the machine where the *lpr* command was invoked).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/l*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpr(1), lpq(1), lpd(1M)

DIAGNOSTICS

lprm: printer: cannot restart printer daemon
The connection to lpd on the local machine failed. This usually means the printer server started at boot time has died or is hung. If it is hung, the master lpd(1M) daemon may have to be killed and a new one started.

BUGS

Since race conditions are possible when updating the lock file, an active job may be incorrectly identified for removal by an *lprm* command issued with no arguments. During the interval between an *lpq(1)* command and the execution of *lprm*, the next job in line may have become active; that job may be removed unintentionally if it is owned by you. To avoid this, supply *lprm* with the job number to remove when a critical job that you own is next in line.

Only a super-user can remove print jobs submitted from another host.

5. lpsched(1M)

lpsched schedules requests taken by *lp(1)* for printing on line printers. See lpmove(1M).

5. lpsetup(1M)

NAME

lpsetup - installs an OPEN 7 printer (GCOS 7 SYSOUT) in the LP spooling system

SYNOPSIS

```
/usr/lib/lpsetup printer
```

DESCRIPTION

lpsetup is used to install a new printer. The *printer* argument is the logical name of the printer for the LP spooling system. It will be used as destination name by further printing requests (see *lp*(1)).

lpsetup is an interactive command.

1) It displays the lists of available interface models and asks you to choose one.

   ```
   MODEL NUMBER:
   ```

   If you press the return key without entering a number, the command asks for the name of an interface program:

   ```
   INTERFACE PROGRAM:
   ```

   You must provide the name of an executable file to be used exclusively for printing on the device associated with the printer.

2) The command asks for the class of the new printer:

   ```
   PRINTER CLASS:
   ```

3) If the model is "Printing on a remote host", the command asks for the name of the remote host:

   ```
   REMOTE HOST:
   ```

   Enter the system name as given in */etc/hosts*.

   The command asks then for the name of the command to be issued on the remote host:

   ```
   COMMAND TO BE ISSUED ON REMOTE HOST (lp, lpr...):
   ```

4) If printing is local, the command asks for the device to be associated with the printer:

   `DEVICE:`

   If the device does not exist, the command asks for the minor number to be used to create the device:

   `MINOR:`

   If you press the return key without entering a number, the command lists the existing sysout or passthrough devices and the minor number they use, before repeating its question.

   If the model is not SYSOUT, the command asks for the DSA address of the printer:

   `PRINTER_SITE:`

   `PRINTER_MAILBOX:`

   The LP administrator may give a list of sites (separated by spaces or tabs) if the physical printer may be accessed by different ways. These ways will be tried in the order they are given, when trying to connect the printer.

5) The command then introduces the new printer in the LP spooling system, connects it, starts the daemon that controls the connection (if the model is neither sysout nor remote printing), and updates */etc/config.lp*.

FILES

`/etc/config.lp` contains the description of the printers to be started by *initlp*(1M).

`/usr/local/<remote_host>_<command>`
contains the interface program created by the command for remote printing on remote_host using command.

SEE ALSO

lp(1), lpstat(1), initlp(1M), lpclean(1M), lpadmin(1M)

5. lpshut(1M)

lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

See lpmove(1M).

5. lpstart(1M), lpstop(1M)

NAME

lpstart, lpstop - start/stop a printer given by its name

SYNOPSIS

```
/usr/lib/lpstart printername
/usr/lib/lpstop printername
```

DESCRIPTION

The *lpstart* command restarts an inactive printer given by a logical name and found with status **off** in the */etc/config.lp* file, inserts this printer in the LP spooling system if not existing and sets its status to **on**.

The *lpstop* command stops an active printer given by a logical name and found with status **on** in the */etc/config.lp* file, disables this printer in the LP spooling system if existing and sets its status to **off**.

The normal sequence of commands to start is:

- search in */etc/config.lp* file a printer whose name matches with *printername* and whose status is **off**.
- extract parameters from this found printer.
- stop the LP scheduler.
- configure the printer in the LP spooling system by the shell procedure */usr/lib/lpinstall*.
- restart the LP scheduler.
- set the printer status to **on** in the */etc/config.lp* file.

The normal sequence of commands to stop is:

- search in */etc/config.lp* file a printer whose name matches with *printername* and whose status is **on**.
- disable the printer in the LP spooling system.
- set the printer status to **off** in the */etc/config.lp* file.

FILES

/etc/config.lp

SEE ALSO

lpadmin(1M), lpdinstall(1M), lpinstall(1M), lplist(1M), lpsched(1M), lpshut(1M), lpstat(1)

5. lpstat(1)

Displays line printer status information (See the *User's Reference Manual*).

5. lpstop(1M)

See lpstart(1M).

5. ls(1)

NAME

ls - list contents of directory

SYNOPSIS

```
ls [-RadCxmlnogrtucpFbqisf] [names]
```

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats:

- the default format is to list one entry per line,
- the -C and -x options enable multi-column formats,
- the -m option enables stream output format.

In order to determine output formats for the -C, -x, and -m options, *ls* uses the COLUMNS environment variable to determine the number of character positions available on one output line. If this variable is not set, the *terminfo(4)* database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The *ls* command has the following options:

-R	Recursively list subdirectories encountered.
-a	List all entries, including those that begin with a dot (.), which are normally not listed.
-d	If an argument is a directory, list only its name (not its contents); often used with -l to get the status of a directory.
-C	Multi-column output with entries sorted down the columns.
-x	Multi-column output with entries sorted across rather than down the page.
-m	Stream output format; files are listed across the page, separated by commas.

-l	List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.
-n	The same as -l, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.
-o	The same as -l, except that the group is not printed.
-g	The same as -l, except that the owner is not printed.
-r	Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
-t	Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See -n and -c.)
-u	Use time of last access instead of last modification for sorting (with the -t option) or printing (with the -l option).
-c	Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (-t) or printing (-l).
-p	Put a slash (/) after each filename if that file is a directory.
-F	Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.
-b	Force printing of non-printable characters to be in the octal \ddd notation.
-q	Force printing of non-printable characters in file names as the character question mark (?).
-i	For each file, print the i-number in the first column of the report.
-s	Give size in blocks, including indirect blocks, for each entry.
-f	Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -l, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.

The mode printed under the -l option consists of ten characters. The first character may be one of the following:

d	the entry is a directory;
b	the entry is a block special file;
c	the entry is a character special file;
p	the entry is a fifo ("named pipe") special file;
-	the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

ls -l Output Example

ls -l (the long list) prints its output as follows:

```
-rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information.

Reading from right to left, you see that the current directory holds one file, named "part2." Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes.

The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2."

Finally, the row of dash and letters tell you that user, group, and others have permissions to read, write, execute "part2."

The execute (x) symbol here occupies the third position of the three-character sequence. A - in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r the file is readable

w the file is writable

x the file is executable

- the indicated permission is not granted

l mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)

s the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on

S undefined bit-state (the set-user-ID bit is on and the user execution bit is off)

t the 1000 (octal) bit, or sticky bit, is on (see *chmod(1)*), and execution is on

T the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than x or -. s also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login."

In the case of the sequence of group permissions, l may occupy the third position. l refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by t or T. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

File's permissions examples:

```
-rwxr--r--                                (1)
-rwsr-xr-x                                (2)
-rw-rwl---                                (3)
```

(1) This example describes a file that is readable, writable, and executable by the user and readable by the group and others.

(2) This example describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

(3) This describes a file that is readable and writable only by the user and the group and can be locked during access.

Command Line Examples

```
ls -a                                                    (1)
ls -aisn                                                 (2)
```

(1) This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

(2) This command will provide you with quite a bit of information including all files, including non-printing ones (a), the i-number-the memory address of the i-node associated with the file-printed in the left-hand column (i); the size (in blocks) of the files, printed in the column to the right of the i-numbers (s); finally, the report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	user IDs for ls -l and ls -o
/etc/group	group IDs for ls -l and ls -g
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

5. lscript(1)

NAME

lscript - translates text files such as source files into POSTSCRIPT files

SYNOPSIS

```
lscript [-f fontname] [-t tabs] [-m margin] [-l lines]
        [-c columns] [-h] [-H] [-T "title"] [-n] [-r rotation]
        [-k filek] [-x filex] [-o fileo] [-L language] files
```

DESCRIPTION

lscript produces a postscript interpretable file from the input files (or standard input) to the standard output.

lscript recognizes the keywords of a source file from the file type, and prints procedural keywords in **bold**, and declarative keywords in ***bold italic***. Commentaries are also recognized and printed in *italic*.

lscript enables you to underline, or use bold and/or italic characters by inserting special sequences. It is also possible to write characters with accents.

POSTCRIPT files can be sent to a POSTCRIPT LASER printer through the standard command *lp*.

OPTIONS

The following options of *lscript* may appear in any order, provided that the names of the files to be printed are all at the end of the command:

-t tabs — *tabs* is an integer value which specifies the tabulation spacing to be used for printing. If not specified, a value of 4 is set for a source file with a language, a value of 8 is set for simple text files.

-f fontname — *fontname* specifies the name or the abbreviation of the postscript font to be used. *fontname* may be selected from the following codes:

C or Courrier
H or Helvetica
A or AvantGarde-Book
HN or Helvetica
B or Bookman-Light
T or Times-Roman
N or NewCenturySchblk-Roman
P or Palatino-Roman
Z or ZapfChancery-Medium-Italic
G or Greek

WARNING

Courrier, which is the default font, is the only font to have a fixed width, therefore the only font you can use for a tabulated file.

-m margin — *margin* is an integer value which specifies a left margin in number of characters. Default is 0.

-l lines — *lines* is the number of significant lines (header excluded) per page. Any value can be used. Default is 78.

-c columns — *columns* is the number of columns for printing. Default is 1.

RESTRICTION

Specifying a number of columns greater than 1 cancels the language processing, and treats files as simple text files.

-h — print with pr-like header.

-H — print with special header.

-T "title" — when an ***-h*** or ***-H*** option is specified, title is put in the header instead of the file name.

-n — print with line numbers.

-r rotation — *rotation* specifies the orientation for printing. Set 0 for portrait printing (default), 90 for landscape printing.

-L language — *language* specifies the language of the source file. Usually the language is retrieved from the name of the file:

xxx.c is taken to be a C-file
xxx.cbl is taken to be a COBOL-file
xxx.f or *xxx.fort* is taken to be a FORTRAN file

When the *-Llanguage* option is specified, the file is processed according to the language specified, which will be one of the following values:

n or No language
C or C
c or COBOL
f or FORTRAN

When the type of the file cannot be determined from its name, and no *-Lx* option is specified, the file is considered as a unique file.

-k filek

Specifies the name of a file which contains words to be considered as keywords, in addition to the language keywords. The file must have the following structure:

$$KEYWORDS:kwd1:kwd2:....:kwdn
$$DEFWORDS:dwd1:dwd2:....:dwdn

The strings located after $$KEYWORDS are processed as procedural keywords, and the strings located after $$DEFWORDS are processed as declarative keywords.

WARNING

The file must begin with $$KEYWORDS. No keyword must appear between $$KEYWORDS and $$DEFWORDS.

The file must not have a declarative word part.

The option ***-k*** *filek* can be used more than once in the command. The keywords are added to the general keywords list.

-x filex

Specifies the name of the file containing words to be considered as keywords, instead of language keywords.

-o fileo

The output is not sent to the printer, but redirected to the file *fileo*.

USAGE

WARNING

The following facilities are restricted to universal texts.

Characters like é, à, ô can be put in the POSTSCRIPT file. Do this by putting the following sequences in the input text for the characters to be printed:

```
- e' for é,
- e` for è,
- a` for à,
- c` for ç,
- e^ for ê,
- o^ for ô,
- u^ for û,
- a^ for â,
- i^ for î,
- i` for ì.
```

The remaining sequences enable you to set a portion of the text as bold, italic, or underlined. In each case, there is a start sequence and an end sequence:

- `[i , [I` for italic,
- `[b , [B` for bold,
- `[u , [U` for underlined.

It is possible to combine two or more of these possibilities for the same text.

If you wish such sequences to be printed rather than interpreted, insert *[c* before the sequence. This is available for both accent sequences and emphasizing sequences.

EXAMPLES

```
lscript -H toto.c
```
generates a postscript listing for a C-file.

```
nroff toto | lscript -c 2 -r 90 -l66 -T"Title of my file"|lp -op
```
prints an *nroff* output in horizontal mode, two *nroff* pages per printer page.

```
lscript -LC -fG tata.cob | lp -op
```
prints the file tata.cob in COBOL mode, in the Greek font.

FILES

/usr/local/lscript/*.kwds standard keyword files

5. ls_attach

NAME

ls_attach - displays list of processes attached to shared memories managed by HSL

SYNOPSIS

```
ls_attach [-n memory_name]
```

DESCRIPTION

This command displays information about the list of processes attached to a specified shared memory (-n option), or on all shared memories. The shared memories are managed by HSL.

Display Example

```
number of shared memories   : 4
number of existing attaches : 8

RON        PROCESS TYPE      pid     ATTACHED TO SHARED MEMORY
-------------------------------------------------------------------------
X1628      OPEN7             120     SUBUX_MEMORY
X1628      OPEN7             170     SUBUX_MEMORY
X2839      GCOS7             ---     SUBUX_MEMORY
X2840      GCOS7             ---     SUBUX_MEMORY
X3137      GCOS7             ---     SUBUX_MEMORY
X1628      OPEN7             130     SUBUX_BUFFER
X1628      OPEN7            1002     EB7V2_MO7
```

The RON column contains the XRON numbers of the GCOS 7 processes which are using the attached shared memories.

The PROCESS TYPE column contains the systems which use the shared memories.

The pid column contains the OPEN 7 process pid which use the shared memories.

The ATTACHED TO SHARED MEMORY column contains the names of the shared memories.

5. ls_shmem

NAME

ls_shmem - displays the list of existing shared memories

SYNOPSIS

```
ls_shmem
```

DESCRIPTION

This command displays the list of existing shared memories, and some information about them.

Display Example

```
number of shared memories   : 4
number of existing attaches : 8

NAME          CREATOR  GROUP     SIZE   ACCESS  CREATED        ATTACHES
-------------------------------------------------------------------------
SUBUX_MEMORY  ROOT     OPERATOR  54844  766     20 Mar 16:45   6
SUBUX_BUFFER  ROOT     OPERATOR  32768  744     20 Mar 16:45   1
EB7V2_MO7     G7EB7    OPERATOR  18416  766     20 Mar 16:45   1
RPC_SH_MEM_01 ROOT     OPERATOR  8192   766     21 Mar 13:32   0
```

5. lsdf(1M)

NAME

lsdf - lists the descriptions of all the disk partitions created by *mkdf*(1M)

SYNOPSIS

```
lsdf [pathname pathname ...]
```

DESCRIPTION

lsdf lists the following details for disk files:

- partition-name,
- gcosfilename,
- media,
- device-class,
- size in blocks,
- major number of the block device,
- major number of the character device,
- minor number of the devices,
- GCOS 7 user,
- GCOS 7 project,
- GCOS 7 billing.

The *pathname* arguments specify the pathname of disk files for which information is to be listed. If *pathname* is absent, the command lists the description of all the partitions under */dev*.

The information about the GCOS 7 file (name, media, device, and size) and the GCOS 7 user (name, project, and billing) are not provided when the command user has no read access right for the OPEN 7 disk device.

The information is listed under the following headings:

```
PARTITION  GCOSFILE  MEDIA  DEVICE  SIZE  MAJ  RMAJ  MIN  USER  PROJ.  BILL.
---------  --------  -----  ------  ----  ---  ----  ---  ----  -----  -----
```

FILES

`/etc/mkdftab`	contains information about the partitions created by mkdf(1M).
`/etc/drivers.tab`	contains information about the drivers.
`/usr/spool/locks/mkdfLOCK`	used to lock /etc/mkdftab.

SEE ALSO

mkdf(1M), mkdfall(1M), rmdf(1M), chdf(1M).

5. lsgfile(1)

NAME

lsgfile - provides a list of the characteristics of all the special files created by the *mkgfile* command.

SYNOPSIS

```
lsgfile [-as] [pathnames]
```

DESCRIPTION

If *-a* is given, the command applies to all the special files created or used by *mkgfile* (1).

Pathnames: each pathname may be a relative or absolute address of:

- a directory: lists the characteristics of all the special files in the directory.
- a special file: lists the characteristics of the specified special file.

If no pathname is given, the command applies to all the special files in the current directory.

The information is listed in the following order:

```
major-number
minor-number
[n][b][h][l]
output-record-length
pathname
gcosfilename[..subfile] type
special file owner's name
special file owner's group
```

If *-s* is given, only the pathname is displayed.

The *n*, *b*, and *l* options, the output record length, and the *gcosfilename* are not indicated when the user issuing the command does not have read rights to the special file.

APPLICATION NOTE

The main lsgfile messages are:

`*** DELETED`	The GCOS7 file does not exist. The GCOS7 file has been deleted or it not already created.
`*** NOT ASSIGNED`	The special file is no longer associated with a GCOS file.
`*** NOT MATCHING`	The OPEN7 file registered in */etc/mkgftab* is not a special file.

FILES

`/etc/mkgftab`

SEE ALSO

cpgtou(1), cputog(1), mkgfile(1), rmgfile(1)

5. lsu, lsuf, lsug

Shell scripts which allow the use of lsufas(1) with metacharacters. See lsufas(1).

5. lsufas(1)

Provides a list of the assignments between a GCOS 7 UFAS file and an OPEN 7 standard file for NFS. These assignments are created by the *mkufas* command.

NAME

lsufas

lsu , *lsuf* , *lsug* are scripts allowing the use of *lsufas* with meta characters.

SYNOPSIS

```
lsufas [-G] [-u pathname]
lsufas [-G] [-g GCOSfilename]
lsufas [-aG]

lsu [pathname]
lsuf [pathname]
lsug [pathname]
```

DESCRIPTION

If *-a* is given, the command applies to all the assignments created by *mkufas* (1). If no pathname is given, nor GCOS filename, the effect is the same as if *-a* is given. In other words, *lsufas* is equivalent to *lsufas -a*.

If *-u* is given, the command applies to the assignment with the OPEN 7 file characterized by *pathname*: this pathname may be relative or absolute.

If *-g* (g in lowercase) is given, the command applies to the assignment with the GCOS 7 file specified.

If *-G* (G in uppercase) is not given, the information is listed in the following order:

- GCOS 7 file name
- OPEN 7 full pathname
- *uid* and *gid* of the user who created the link by *mkufas*
- type of the UFAS file (SEQ or REL)
- file state, which may be, according to the access:
  - CLOSED
  - READ
  - WRITE

If *-G* is given, the following information only is displayed:

- GCOS 7 file name
- OPEN 7 base name
- type of the UFAS file (SEQ or REL)
- state of the file.

Meta characters are not allowed in this command, but there are 3 shell-scripts which do allow them:

```
lsu
lsuf
lsug
```

The command *lsu [pathname]* is equivalent to the *lsufas -u pathname*, but with meta characters allowed in *pathname*. If no assignment is found corresponding to *pathname*, no error message is displayed. If no pathname is given (for example, *lsu*) the command is equivalent to *lsufas -a*.

EXAMPLE:

```
lsu *
```
gives all the assignments with OPEN 7 files in the current directory.

The command *lsuf* is the same as *lsu*, except that:

- if only one file is characterized by pathname,
- and if this file is not assigned to a GCOS 7 file,

an error message is displayed.

The command *lsug* has the same effect as *lsu*, but the information is displayed according to the *-G* option of *lsufas*.

SEE ALSO

mkufas(1), deaufas(1)

APPLICATION NOTE

UNIX size is always a multiple of the UFAS record size.

6. Commands Beginning with the Letter "M"

6. m4(1P)

NAME

m4 - macro processor, preprocesses files, expanding macro definitions

SYNOPSIS

```
m4 [options] [files]
```

DESCRIPTION

The *m4* command is a macro processor intended as a front end for C and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

-e	Operate interactively. Interrupts are ignored and the output is unbuffered.
-s	Enable line sync output for the C preprocessor (#line ...)
-Bint	Change the size of the push-back and argument collection buffers from the default of 4,096.
-Hint	Change the size of the symbol table hash array from the default of 199. The size should be prime.
-Sint	Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
-Tint	Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any -D or -U flags:

-Dname[=val]	Defines name to val or to null in val's absence.
-Uname	undefines name.

Macro calls

Macro calls have the form:

```
name(arg1,arg2, ..., argn)
```

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines 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 right 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. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

Built-in Macros

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	the second argument is installed as the value of the macro whose name is the first argument. 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; $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotes).
undefine	removes the definition of the macro named in its argument.
defn	returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	like define, but saves any previous definition.
popdef	removes current definition of its argument(s), exposing the previous one, if any.
ifdef	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. The word *unix* is predefined on system versions of *m4*.

shift	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	changes quote symbols to the first and second arguments. The symbols may be up to five characters long. *changequote* without arguments restores the original values (i.e., `').
changecom	change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	*m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; 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 is discarded.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next new-line.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. 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	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	returns the value of its argument decremented by 1.

eval	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; 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	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to include, except that it says nothing if the file is inaccessible.
syscmd	executes the system command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap('cleanup()')
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by traceon can be untraced only by specific calls to traceoff.

SEE ALSO

cc(1), cpp(1).

6. make(1P)

NAME

make - maintain, update, and regenerate groups of programs

SYNOPSIS

```
make [-f makefile] [-p] [-i] [-k] [-s] [-r] [-n] [-b] [-e]
[-u] [-t] [-q] [names]
```

DESCRIPTION

The *make* command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

-f makefile
Description filename. *makefile* is assumed to be the name of a description file.

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

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

-k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

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

-r Do not use the built-in rules.

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

-b Compatibility mode for old makefiles.

-e Environment variables override assignments within makefiles.

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

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

.DEFAULT
If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.

.PRECIOUS
Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT
Same effect as the -s option.

.IGNORE
Same effect as the -i option.

make executes commands in *makefile* to update one or more target names. Name is typically a program. If no -f option is present, *makefile*, *Makefile*, and the Source Code Control System (SCCS) files *s.makefile*, and *s.Makefile* are tried in order. If makefile is -, the standard input is taken. More than one -f makefile argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

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

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

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

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

Command lines are executed one at a time, each by its own shell. The SHELL environment variable can be used to specify which shell make should use to execute commands. The default is /bin/sh. The first one or two characters in a command can be the following: -, @, -@, or @-. If @ is present, printing of the command is suppressed. If - is present, make ignores an error. A line is printed when it is executed unless the -s option is present, or the entry .SILENT: is in makefile, or unless the initial character sequence contains a @. The -n option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the MAKEFLAGS macro under Environment). The -t (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate make. If the -i option is present, or the entry .IGNORE: appears in makefile, or the initial character sequence of the command contains -, the error is ignored. If the -k option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The -b option allows old makefiles (those written for the old version of make) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name .PRECIOUS.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The -e option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built- in rules.

The MAKEFLAGS environment variable is processed by make as containing any legal input option (except -f and -p) defined for the command line. Further, upon invocation, make ``invents'' the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for ``super-makes''. In fact, as noted above, when the -n option is used, the command $(MAKE) is executed anyway; hence, one can perform a make -n recursively on a whole software system to see what would have been executed. This is because the -n is put in MAKEFLAGS and passed to further invocations of $(MAKE). This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string include appears as the first seven letters of a line in a makefile, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros.

Macros

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

Another special macro is VPATH. The VPATH macro should be set to a list of directories separated by colons. When make searches for a file as a result of a dependency relation, it will first search the current directory and then each of the directories on the VPATH list. If the file is found, the actual path to the file will be used, rather than just the filename. If VPATH is not defined, then only the current directory is searched.

VPATH can be used when one has several programs that compile from the same source. The source can be kept in one directory and each set of object files (along with a separate makefile) would be in a separate subdirectory. The VPATH macro would point to the source directory in this case.

Internal Macros

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

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

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

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

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

or:

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

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

$% The $% macro is only evaluated when the target is an archive library member of the form lib(file.o). In this case, $@ evaluates to lib and $% evaluates to the library member, file.o.

Four of the five macros can have alternative forms. When an upper case D or F is appended to any of the four macros, the meaning is changed to ``directory part'' for D

and ``file part'' for F. Thus, $(@D) refers to the directory part of the string $@. If there is no directory part, ./ is generated. The only macro excluded from this alternative form is $?.

Suffixes

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

```
.c .c~ .f .f~ .sh .sh~
.c.o .c.a .c~.o .c~.c .c~.a
.f.o .f.a .f~.o .f~.f .f~.a
.h~.h .s.o .s~.o .s~.s .s~.a .sh~.sh
.l.o .l.c .l~.o .l~.l .l~.c
.y.o .y.c .y~.o .y~.y .y~.c
```

The internal rules for make are contained in the source file rules.c for the make program. These rules can be locally modified. To print out the rules compiled into the make on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file [see sccsfile(4)]. Thus, the rule .c~.o would transform an SCCS C source file into an object file (.o). Because the s. of the SCCS files is a prefix, it is incompatible with make's suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., .c:) is the definition of how to build x from x.c. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for .SUFFIXES. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f
.f~
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; .SUFFIXES: with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

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

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

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, CFLAGS, LFLAGS, and YFLAGS are used for compiler options to cc(1P), lex(1P), and yacc(1P), respectively. Again, the previous method for examining the current rules is recommended.

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

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus lib(file.o) and $(LIB)(file.o) both refer to an archive library that contains file.o. (This assumes the LIB macro has been previously defined.) The expression $(LIB)(file1.o file2.o) is not legal. Rules pertaining to archive libraries have the form .XX.a where the XX is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the XX to be different from the suffix of the archive member. Thus, one cannot have lib(file.o) depend upon file.o explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
   @echo lib is now up-to-date
.c.a:
   $(CC) -c $(CFLAGS) $<
   $(AR) $(ARFLAGS) $@ $*.o
   rm -f $*.o
```

In fact, the .c.a rule listed above is built into make and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
   $(CC) -c $(CFLAGS) $(?:.o=.c)
   $(AR) $(ARFLAGS) lib $?
   rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The $? list is defined to be the set of object filenames (inside lib) whose C source files are out-of-date. The substitution mode translates the .o to .c. (Unfortunately, one cannot as yet transform to .c~; however, this may become possible in the future.) Note also, the disabling of the .c.a: rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

APPLICATION USAGE

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

Filenames with the characters = : @ will not work. Commands that are directly executed by the shell, notably cd(1), are ineffectual across new-lines in make. The syntax (lib(file1.o file2.o file3.o) is illegal. You cannot build lib(file.o) from file.o. The macro $(a:.o=.c~) does not work. Named pipes are not handled well.

FILES

```
[Mm]akefile and s.[Mm]akefile
/bin/sh
```

SEE ALSO

cc(1P), cd(1), lex(1P), sh(1), yacc(1P), printf(3S), sccsfile(4).

6. master(4)

Master device information table. Referred to in sysdef(1M). See the *Programmer's Reference Manual*.

6. mem(7), kmem(7)

Standard UNIX kernel memory driver adapted to DPS 7000, giving access to the kernel memory.

Several UNIX commands read information directly from the kernel memory using:

- kmem driver
- */unix* file which holds UNIX kernel binary code and especially its symbol table of structures in memory.

Example:

ps	processes running and their status
crash	kernel image analyzer

NAME

mem, kmem

DESCRIPTION

/dev/mem is not implemented.

/dev/kmem is a special file which is an image of the kernel system virtual memory. It may be used, for example, to examine the system.

No *ioctl* (2) system calls are available on these devices.

SEE ALSO

crash(1M)

6. mesg(1)

NAME

mesg - permit or deny messages

SYNOPSIS

```
mesg [-n] [-y]
```

DESCRIPTION

mesg with argument *n* forbids messages via *write(1)* by revoking non-user write permission on the user's terminal.

mesg with argument *y* reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

NOTES

C2 Secure Environment
In a Secure environment, this command can be used only by the user with terminal subsystem privilege.

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

6. mkdf(1M)

NAME

mkdf - creates a new disk partition

SYNOPSIS

```
mkdf [-m media] [-s size] [-U sizeunit] [-f gcosfilename]
     [-p gcospassword] [ -u gcosuser] [-k gcosbilling]
     [-j gcosproject]] [-i minor] filename
```

DESCRIPTION

mkdf creates a cataloged GCOS 7 file and formats it as an OPEN 7 volume partition. Two inodes are created in the */dev* directory for the partition: e.g., if filename is USERSF, a block special file */dev/USERSF* and a character special file */dev/rUSERSF*.

The options and arguments are:

filename	specifies the name of the partition to be created (1 to 13 uppercase characters or digits).
-f *gcosfilename*	specifies the name of the GCOS 7 file to be created (or used if the ***-s*** option is absent). If ***-f*** *gcosfilename* is absent, *mkdf* builds the name of the GCOS 7 file from the master directory which contains the root filesystem and from the partition name; for example, for a USERSF partition and a root filesystem on OPEN 7.VOL.ROOT, the GCOS 7 file name is OPEN 7.USERSF.
-m *media*	specifies the name of the media on which the GCOS 7 file is to be created (HIT81 for example); this parameter is mandatory if ***-s*** is given.
-s *size*	specifies the size of the partition to be created. Depending on the **-U** argument, it is specified as a number of blocks (default) or a number of cylinders that will make up the partition. If this option is not given, the *mkdf* command assumes that the GCOS 7 file already exists and it creates and assigns the special files only.
-U *sizeunit*	specifies the unit of the ***-s*** argument. Allowed values are "block" (default) and "cyl" (for cylinders). This parameter is forbidden when ***-s*** is not given.
-p *gcospassword*	specifies the GCOS 7 user's password to be checked for access rights to the GCOS 7 file; if ***-p*** *gcospassword* is not given, the rights are those of the user who started OPEN 7.

-u *gcosuser* specifies the GCOS 7 user name to be checked for access rights to the GCOS 7 file.

-k *gcosbilling* specifies the GCOS 7 billing to be checked for access rights to the GCOS 7 file; if ***-k*** *gcosbilling* is not given, the GCOS 7 user's default billing is used.

-j *gcosproject* specifies the GCOS 7 project to be checked for access rights to the GCOS 7 file; if ***-j*** *gcosproject* is not given, the GCOS 7 user's default project is used.

-i *minor* specifies the minor number to be used for the creation of the inodes; if this parameter is not given, *mkdf* searches for a free minor number.

APPLICATION NOTES

The command performs the following operations:

- creates the special files if they do not exist;
- if ***-s*** size is given:
  - allocates the GCOS 7 file,
  - formats the GCOS 7 file as an OPEN 7 volume,
  - assigns the GCOS 7 file to the special files,
  - creates the directory *lost+found*;
- otherwise (GCOS 7 file is presumed to exist):
  - assigns the GCOS 7 file to the special files,
  - checks if the GCOS 7 file is a valid OPEN 7 volume,
  - if the check fails, issues a warning message but does not remove assignment (to enable further run of *fsck(1M)*);
- updates /*etc/mkdftab*.

FILES

`/etc/dsktab`	contains the description of disks (media, device-class, blocks/cylinders, rotational gap size). It must be updated by the OPEN 7 administrator.
`/etc/mkdftab`	contains the list of the filenames created by *mkdf*.
`/dev/u_mkdf and /mnt_mkdf`	are used to create and/or to check the GCOS 7 file.
`/etc/drivers.tab`	contains information about the drivers.
`/usr/spool/locks/mkdflock`	is used to lock */etc/mkdftab*.

SEE ALSO

mkfs(1M), lsdf(1M), rmdf(1M), chdf(1M), mkdfall(1M).

6. mkdfall(1M)

NAME

mkfdfall - assigns multiple disk partitions

SYNOPSIS

```
/etc/mkdfall
```

DESCRIPTION

This command may be executed only by the superuser.

mkdfall is used to assign GCOS 7 files formatted as OPEN 7 partitions to block special devices specified in */etc/mkdftab*: it calls *mkdf(1M)* for each line of */etc/mkdftab*.

FILES

`/etc/mkdftab` — updated by *mkdf*(1M), *chdf*(1M) and *rmdf*(1M) and consulted by *lsdf*(1M).

field 1: partition name
field 2: GCOS 7 file name
field 3: media
field 4: device-class
field 5: size in blocks
field 6: encrypted GCOS 7 password
field 7: GCOS 7 user name
field 8: GCOS 7 project
field 9: GCOS 7 billing

Fields are separated by spaces. Fields 6 to 9 are optional fields, fields 3, 4, 6, 7, 8, and 9 are set to ? (question mark) when they are unknown.

`/etc/drivers.tab` — contains information about the drivers.

`/usr/spool/locks/mkdfLOCK` — used to lock /etc/mkdftab.

SEE ALSO

lsdf(1M), mkdf(1M), chdf(1M), rmdf(1M).

6. mkdftab(4)

Specifies special devices. Updated by mkdf(1M), chdf(1M) and rmdf(1M). Its contents are displayed by lsdf(1M). See mkdfall(1M)

6. mkdir(1)

NAME

mkdir - make directories

SYNOPSIS

```
mkdir [-m mode]  [-p] dirname ...
```

DESCRIPTION

mkdir creates the named directories in mode 0777 (possibly altered by *umask(1)*).

Standard entries in a directory (e.g. the files, . for the directory itself, and .. for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory. The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

-m	This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod(1)*.
-p	With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure ltr/jd/jan, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

sh(1), rm(1), umask(1).
intro(2), mkdir(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

mkdir returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero.

6. mkfs(1M)

NAME

mkfs - constructs a filesystem

SYNOPSIS

```
/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]
```

DESCRIPTION

mkfs constructs a filesystem by writing to the *special* file, using the values found in the remaining arguments of the command line.

The command indicates on which volume it is going to work by printing the name of the GCOS 7 file corresponding to *special*, then waits 10 seconds before starting to construct the filesystem. During this 10-second pause the command can be aborted by entering a delete (DEL).

If the second argument is a numeric string, the size of the filesystem is the value in blocks expressed as a decimal number. This is the number of physical (4096-byte) disk blocks the filesystem will occupy. If the optional number of i-nodes is not given, the default is the number of logical blocks divided by 4. *mkfs* builds a filesystem with a single empty directory. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and takes its instructions from that file. The prototype file contains tokens separated by spaces or new-line characters. The first token is the name of a file to be copied onto block zero as the bootstrap program (see *boot*(8)). The second token is a number specifying the size of the created filesystem in physical disk blocks. The next token is the number of inodes in the filesystem. The maximum number of i-nodes configurable is 65500. The next set of tokens gives the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6-character string. The first character specifies the type of the file. (The characters - *b c d* specify regular, block special, character special, and directory file respectively.) The second character of the type is either *u* or - to specify *set-userid* mode or not. The third is *g* or - for *set-group-id* mode or not. The rest of the mode is a 3 digit octal number giving the owner and group, and other read, write, execute permissions (see *chmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token is a pathname of which the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries and then reads a list of names and (recursively) files specifications for the entries in the directory. The scan is terminated with the token $.

A sample prototype specification follows:

```
/etc/diskboot
13776 1500
d-777 3 1
usr d--777 3 1
 sh ---755 3 1 /bin/sh
 ken d--755 6 1
  $
 b0 b--644 3 1 0 0
 c0 c--644 3 1 0 0
 $
$
```

The final arguments in both command syntaxes specify the rotational gap and the number of blocks per cylinder, specified in 4096-byte blocks. If the gap and blocks/cylinder are not specified or are considered illegal values, a default value is used.

RESTRICTIONS

If a prototype is used, it is not possible to initialize a file larger than 64 Kbytes, nor is there a way to specify links.

SEE ALSO

chmod(1), dir(4), fs(4).

6. mkgfile(1)

NAME

mkgfile - creates an OPEN 7 special character file and associates it with a GCOS 7 file description

SYNOPSIS

for SL library:

```
mkgfile -s subfile [-t type] [-p gcospassword [-u gcosuser]
        [-k gcosbilling] [-j gcosproject]] [-r nnn] [-h] [-l]
        [-n] [-b] [-B] [-F] [-M drivername] [-i minor]
        pathname gcosfilename
```

for UFAS file:

```
mkgfile [-p gcospassword [-u gcosuser] [-k gcosbilling]
        [-j gcosproject]] [-r nnn] [-n] [-b] [-B] [-F]
        [-L recmaxlg -C cisize -S filesize  -I incrsize
         -U allocation_unit]  [-M drivername] [-i minor]
         -D UFAS pathname gcosfilename
```

DESCRIPTION

mkgfile enables you to associate a GCOS 7 file with an OPEN 7 special character file and authorizes OPEN 7 commands or programs to access the GCOS 7 file.

The GCOS 7 file may be a source library subfile or a sequential UFAS file.

The options and arguments are:

pathname	specifies the special file name to be associated with the GCOS 7 file. If the special file does not exist, the *mkgfile* command creates it, using the *-i* argument when given or looking for a free minor number when *-i* is not provided.
gcosfilename	is a valid GCOS 7 file name which must be a cataloged file name. If the GCOS 7 file is a UFAS file which does not exist, *gcosfilename* must contain the device and media names (GCL syntax), and the GCOS 7 file will then be created at the first OPEN for writing in the special file.
-s subfile	is the subfile member name when the GCOS 7 file is a library.

-t type

is the type of the subfile when the GCOS 7 file is an SL library. The possible values are:

`CMD` for GCL language file
`CL` for C language file
`ADL` for Application Definition Language file
`COB` for COBOL language file
`CBX` for COBOL language file
`DAT` for Data file
`EDT` for Edited file
`FOR` for FORTRAN language file
`GPL` for GPL language file
`JCL` for JCL language file
`LL` for LE LISP language file
`PAS` for PASCAL language file

The default type is DAT. Type can be given in lower or upper case. This option applies only to the -s option.

-p gcospassword

specifies the GCOS 7 password to be checked for access rights to the GCOS 7 file; if *-p gcospassword* is not given, the rights are those of the OPEN 7 project.

-u gcosuser

specifies the GCOS 7 user name to be checked for access rights to the GCOS 7 file; if *-u gcosuser* is not given, the USER environment variable is used.

-k gcosbilling

specifies the GCOS 7 billing to be checked for access rights to the GCOS 7 file; if *-k gcosbilling* is not given, the user's default billing is used.

-j gcosproject

specifies the GCOS 7 project to be checked for access rights to the GCOS 7 file; if -j gcosproject is not given, the user's default project is used.

-r nnn

specifies the output record length (default 0) when transferring data from OPEN 7 to GCOS 7.

If nnn != 0 then the input records are split or space-filled to the correct length; if nnn=0 then the input record length is maintained.

-n

specifies fixed length records. When this option is present, line-feed characters are maintained when transferring data from OPEN 7 to GCOS 7 ; when it is not present, line-feed characters are considered as record separators and cancelled.

If *-n* is present, when transferring data from GCOS 7 to OPEN 7, no line-feed character is added to the end of output records and trailing spaces are not removed.

If *-n* is present, the *-r* argument must be non-zero.
To avoid unpredictable results, -n and *-r nnn* options must be used when you are transferring binary files.

Variable-length record transfer must be used only with pure text files (7-bit ASCII). The maximum record size of the

GCOS 7 file is important. When the data is greater than the record size, the output is split into several records. When transferring data from GCOS 7 to OPEN 7, for each record, trailing spaces are deleted and a line feed (LF) character is added.

-h — specifies SARF format. (only with *-s* option (library subfile)).

Without this option, when transferring data from GCOS 7 to OPEN 7, the SSF header and the control records of the subfile are suppressed; when *-h* is present, they are maintained.

With the *-h* option, when transferring data from OPEN 7 to GCOS 7, the subfile is written in SARF format ; without the *-h* option, SSF format is used.

-l — specifies presence of line numbers (only with the *-s* option (library subfile)).

When transferring data from GCOS 7 to OPEN 7, the line number is read in the SSF header of each record and inserted at the beginning of each line on 8 characters.

When transferring data from OPEN 7 to GCOS 7 the line number is taken in the first 8 characters of each line and written in the SSF header of each record. The 8 characters are subsequently ignored.

The *-l* and *-h* options are mutually exclusive.

-b — no EBCDIC/ASCII translation; this option may be used only if the *-n* option is present.

-B — causes spaces at the end of a record to be taken into account.

-F — raw mode; each write/read in the special file immediately maps a write/read of a record in the GCOS 7 file.

The *-F* option is used by an application to read/write any GCOS 7 file (queued files or UFAS files).

In read mode, each *read*(2) returns the length of the data, and the actual data.

In write mode, a record is written to the GCOS 7 file with the length of *write*(2) nbbytes.

-L recmaxlg — specifies the maximum record length in bytes of the UFAS file. It must be greater than, or equal to the value of the *-r* argument and it is used when the GCOS 7 file is a UFAS file (option *-D UFAS*) which does not exist.

-C cisize	specifies the Control Interval length in bytes of the UFAS file. It is used when the GCOS 7 file is a UFAS file (option *-D UFAS*) that does not exist.
-S filesize	Specifies the initial length of the UFAS file (the unit is given by the *-U* argument). It is used when the GCOS 7 file is an UFAS file (option *-D UFAS*) that does not exist.
-I incrsize	specifies the increment length of the UFAS file (the unit is given by the *-U* argument). It is used when the GCOS 7 file is a UFAS file (option *-D UFAS*) which does not exist.
-U allocation_unit	specifies the unit of the *-S* and *-I* arguments and may be CYL (cylinder), TRK (track) or 32K (32kbytes). It is used when the GCOS 7 file is a UFAS file (option *-D UFAS*) which does not exist.
-i minor	the minor number associated with the OPEN 7 special file. If the special file exists, the *-i* argument is used for checking, otherwise it is used to create the special file (see pathname).
-D symbol	specifies the type of the GCOS 7 file and may be: SL or sl: GCOS 7 file is a source library member. UFAS or ufas: GCOS 7 file is a UFAS file.
-M drivername	specifies the driver to be used for access to the GCOS 7 file (see Application Notes below) and may be: QFMGT or qfmgt quick driver, (default) FMGT or fmgt complete driver.

The *-D* and *-s* arguments must be compatible when given together.

EXAMPLES

Using the command *grep* on the subfile *program_c* of the SL library PROJ.SLLIB whose owner is the default project of the GCOS 7 user MY_NAME:

```
mkgfile -s program_c -u MY_NAME -p MY_PASS progc PROJ.SLLIB

grep main progc
```

APPLICATION NOTES

The *mkgfile* command creates the OPEN 7 special file but not the associated GCOS 7 file. The only errors that the *mkgfile* command can return is the lack of free entry point in the QFMGT driver ("no more free minor" message). All errors about the GCOS 7 file (not present, no access...) are returned after the first access to this driver or after the next command such as *lsgfile*, *rmgfile*.

The **first access** after the *mkgfile* command asks the GCOS 7 catalog facility to check for access rights to the GCOS 7 file (if the *-p* option is present) and updates the driver table.

However, the effective assignment (GPL primitives H_CRFD and H_ASSIGN GCOS 7) of the special file to the GCOS 7 file only occurs when the special file is opened. The GCOS 7 file is deassigned (GCOS 7 GPL primitives H_DEASSIGN and H_DLFD) when the special file is closed.

WARNING

You cannot create a file in a GCOS7 directory of which you are not the OWNER, even if you specify the user name and password of the directory user.

Ignore this warning if you set the GCOS7 patch O5954.01

The QFMGT driver has restricted functions: a process that has opened a QFMGT special file may not fork (*fork*(2) fails with *errno* set to ENOFORK).

For example, if *xxx* special file has been created using the command:

```
mkgfile -s xxx_member xxx OPEN7.SLLIB
```

The following shell command

```
for i in 1 2 3 4 5
do
cat /etc/passwd
done > xxx
```

will fail with the message:

```
sh: cannot fork because of a direct call GCOS 7 driver
```

But the QFMGT driver is faster than the FMGT driver and its use improves the performance of file transfer applications. It is used by the *cputog*(1) and *cpgtou* (1) commands, and should be chosen when the special file created by *mkgfile* is going to be used in an FTP session.

When the you associate a GCOS 7 file and the OPEN 7 special file through the *gmenu* command (standard method described in the *OPEN 7 User's Guide*), *gmenu* performs an *mkgfile* with QFMGT.

FILES

`/etc/mkgftab`	contains the description of the special files created by *mkgfile*.
`/etc/drivers/tab`	contains information about the drivers.
`/usr/spool/locks/mkgfLOCKS`	is used to lock */etc/mkgftab*.

SEE ALSO

cpgtou(1), cputog(1), lsgfile(1), rmgfile(1)

6. mkhosts(1M)

NAME

mkhosts - make node name commands

SYNOPSIS

`/etc/mkhosts`

DESCRIPTION

mkhosts makes the simplified forms of the *rshell(1)* and *rlogin(1)* commands.

For each node listed in */etc/hosts*, *mkhosts* creates a link to */usr/bin/rshell* in */usr/hosts*.

Each link's name is the same as the node's official name in */etc/hosts*.

SEE ALSO

rshell(1), rlogin(1).

6. mklost+found

NAME

mklost+found - creates a lost and found directory for the fsck command.

SYNOPSIS

```
mklost+found
```

DESCRIPTION

The *mklost+found* command creates a lost and found directory in the current directory. A number of empty files are created within the lost and found directory and then removed so that there are empty slots for the *fsck* command. The *fsck* command reconnects any orphaned files and directories by placing them in the lost and found directory with an assigned i-node number.

The *mklost+found* command is not normally needed, since the *fsck* command automatically creates the lost and found directory when a new file system is created.

FILES

/usr/sbin/mklost+found pathname of the *mklost+found* command.

SEE ALSO

fsck(1M), mkfs(1M).

6. mknod(1M)

NAME

mknod - creates a special file

SYNOPSIS

```
/etc/mknod name c|b major minor
/etc/mknod name p
```

DESCRIPTION

mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry.

In the first case, the second is b if the special file is block-type (disks, tape) or c if it is character-type (other devices).

The last two arguments are numbers specifying the *major* device type and the minor device (e.g., unit, drive, or line number), which may be either decimal or octal (an octal number must be prefixed with a zero).

The assignment of major device numbers is specific to each system. See the tables below.

mknod can also be used to create fifo's (a.k.a named pipes) (second case in *SYNOPSIS* above).

Major Device Numbers for Block-type Special Files

Device	Major Number
OPEN 7 disk	0
rfu for ramdisk	1
Magnetic tape	2

Major Device Numbers for Character-type Special Files

Device	Major Number
Console	0
tty vcam driver	1
tty owner	2
line printer	3
physical and virtual memory	4
Null driver	5
CTMS driver	6
UNIX volume	7
exabyte driver	8
magnetic tape driver	9
magnetic tape no wait	11
PT driver	12
kernel profiler	13
pty slave side	14
pty controller	15
UFAS file	16
syslog driver	18
lan driver	19
fmgt	21
tcptty driver	22
XTI	23
log stream	24
stcp stream	25
sudp stream	26
ptq driver	27, 28, 29, 30, 31, 32, 33, 34
sgxti stream driver	35, 36, 37, 38

6. mkufas(1)

NAME

mkufas - associates an OPEN 7 standard file with a GCOS 7 UFAS file

SYNOPSIS

```
mkufas  [-T ufastype] [-p gcospassword [-u gcosuser]
        [-k gcosbilling] [-j gcosproject]] [-b]
        [-U allocation_unit] [-L recmaxlg -C cisize -S filesize
         -I incrsize] pathname gcosfilename
```

DESCRIPTION

mkufas associates a GCOS 7 file with an OPEN 7 standard file and authorizes access to the GCOS 7 file via OPEN 7 commands or programs. The GCOS 7 file may be a sequential or relative UFAS file. If one of the files does not exists, the command creates it.

The options and arguments are:

pathname	The OPEN 7 file name to be associated with the GCOS 7 file. If this file does not exist, *mkufas* creates it.
gcosfilename	A valid GCOS 7 file name which must be a cataloged file name. The GCOS 7 file is a UFAS file, relative or sequential; if it does not exist, *gcosfilename* must contain the devclass and media names (GCL syntax), and this file will then be created at the first OPEN.
-T ufastype	The type of the GCOS 7 UFAS file. The possible values are: `SEQ` or `seq` for sequential `REL` or `rel` for relative The default type is SEQ.
-p gcospassword	The GCOS 7 password to be checked for access rights to the GCOS 7 file; if *-p gcospassword* is not given, the rights are those of the OPEN 7 project.
-u gcosuser	The GCOS 7 user name to be checked for access rights to the GCOS 7 file; if *-u gcosuser* is not given, the USER environment variable is used.
-k gcosbilling	The GCOS 7 billing to be checked for access rights to the GCOS 7 file; if *-k gcosbilling* is not given, the user's default billing is used.
-j gcosproject	The GCOS 7 project to be checked for access rights to the GCOS 7 file; if *-j gcosproject* is not given, the user's default project is used.

-b	No EBCDIC/ASCII translation.
-L recmaxlg	The maximum record length in bytes of the UFAS file; used when the GCOS 7 UFAS file does not exist.
-C cisize	The Control Interval length in bytes of the UFAS file; used when the GCOS 7 UFAS file does not exist.
-S filesize	The initial length of the UFAS file (the unit is given by the -U argument); used when the GCOS 7 UFAS file does not exist.
-I incrsize	The increment length of the UFAS file (the unit is given by the -U argument); used when the GCOS 7 UFAS file does not exist.
-U allocation_unit	The unit of the -S and -I arguments and may be CYL (cylinder), TRK (track), or 32K (32 Kbytes); used when the GCOS 7 UFAS file does not exist. The default value is CYL.

APPLICATION NOTES

mkufas asks the GCOS 7 catalog facility to check for access rights to the GCOS 7 file (if the -p option is present) and updates the driver table.

The OPEN 7 standard file is actually assigned to the GCOS 7 UFAS file only when the file is opened (GPL primitives H_CRFD and H_ASSIGN). The GCOS 7 file is deassigned (GPL primitives H_DEASSIGN and H_DLFD) when closed.

WARNING

You cannot create a file in a GCOS7 directory of which you are not the OWNER, even if you specify the user name and password of the user.

Ignore this warning if you set the GCOS7 patch O5954.01

SEE ALSO

lsufas(1), deaufas(1)

Section 7.4, *NFS Administration* in the *OPEN 7 Administrator's Guide* gives information about system calls and commands over NFS/UFAS.

6. more(1)

Displays the contents of files one screen at a time (See the *User's Guide*).

6. mount(1M)

NAME

mount - mounts a filesystem

SYNOPSIS

```
/etc/mount [ -p ]
/etc/mount -a[fnv] [ -t type ]
/etc/mount [-f nrv] [-t type] [-o options] filesystem directory
/etc/mount [-v fn] [-o options] filesystem | directory
```

DESCRIPTION

mount attaches a named filesystem to the filesystem hierarchy at the pathname location *directory*, which must already exist. If *directory* has any contents prior to the mount operation, these remain hidden until the filesystem is unmounted. If *filesystem* is of the form *host:pathname*, it is assumed to be an NFS filesystem (type nfs).

mount maintains a table of mounted filesystems in */etc/mtab*, described in *fstab*(4). If executed without an argument, mount displays the contents of this table, and for each filesystem, displays the name of the GCOS file assigned to it. If executed with either a filesystem or directory only, *mount* searches the file */etc/fstab* for a matching entry, and mounts the filesystem indicated in that entry on the indicated directory.

MOUNT OPTIONS

-p Prints the list of mounted filesystems in a format suitable for use in */etc/fstab*.

-a All. Attempts to mount all the filesystems described in */etc/fstab*. If a type argument is specified with ***-t***, mounts all filesystems of that type.
Using ***-a***, *mount* builds a dependency tree of mount points in */etc/fstab*. *mount* will correctly mount these filesystems regardless of their order in */etc/fstab* (except loopback mounts).

-f Fakes a */etc/mtab* entry, but does not actually mount any filesystems.

-n Mounts the filesystem without making an entry in */etc/mtab*.

-v Verbose. Displays a message indicating each filesystem being mounted.

-t type Specifies a filesystem type. The accepted types are *sysV* and *nfs*. See *fstab*(5) for a description of these types.

-r Mounts the specified filesystem as read only, even if the entry in */etc/fstab* specifies that it is to be mounted read/write.

Physically write-protected and magnetic tape filesystems must be mounted read only, otherwise errors occur when the system attempts to update access times, even if no write operation is attempted.

-o options Specifies filesystem options - a list of comma-separated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

options valid on all filesystems:

rw|ro
Read/write or read only.
The default is *rw,suid*

options specific to nfs (NFS) filesystems:

bg|fg
If the first attempt fails, retry in the background (*bg*) or in the foreground (*fg*).

retry=n
The number of times to retry the mount operation.

rsize=n
Sets the read buffer size to n bytes.

wsize=n
Sets the write buffer size to n bytes.

timeo=n
Sets the NFS timeout to n tenths of a second.

retrans=n
The number of NFS retransmissions.

port=n
The server IP port number.

soft|hard
Returns an error if the server does not respond (*soft*), or continues the retry request until the server responds (*hard*).

intr
: Allows keyboard interrupts on hard mounts.

acregmin=n
: Holds cached attributes for at least n seconds after file modification.

acregmax=n
: Holds cached attributes for no more than n seconds after file modification.

acdirmin=n
: Holds cached attributes for at least n seconds after directory update.

acdirmax=n
: Holds cached attributes for no more than n seconds after directory update.

actimeo=n
: Sets minimum and maximum times for regular files and directories to n seconds.

Standard defaults are:

```
fg,retry=10000,timeo=11,retrans=5,port=NFS_PORT,hard,\

acregmin=3,acregmax=60,acdirmin=30,acdirmax=60
```

actimeo has no default; it sets *acregmin*, *acregmax*, *acdirmin*, and *acdirmax*.

Defaults for *rsize* and *wsize* are set internally by the system kernel.

NFS FILESYSTEMS

Background and Foreground

File systems mounted with the *bg* option indicate that *mount* is to retry in the background if the server's mount daemon (*mountd*(1M)) does not respond. *mount* retries the request up to the count specified in the *retry=n* option.

Once the file system is mounted, each NFS request made in the kernel waits *timeo=n* tenths of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When the number of retransmissions has reached the number specified in the *retrans=n* option, a file system mounted with the *soft* option returns an error on the request, whereas a file system mounted with the *hard* option prints a warning message and continues to retry the request.

Read/Write and Read Only

Filesystems that are mounted *rw* (read/write) should use the *hard* option.

Interrupting Processes With Pending NFS Requests

The *intr* option allows keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted filesystem.

File Attributes

The attribute cache retains file attributes on the client.

Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories.

Setting *actimeo=n* extends flush time by n seconds for both standard files and directories.

NFS7/UFAS filesytems

Client machines mounting OPEN 7 directories containing UFAS files must take the following into account.

First, all the options available for the *mount*(1M) command are strictly for NFS7 filesystems only. However, because these are treated locally (client side), they are still valid for NFS7/UFAS. Nevertheless, the options *timeo* (timeout) and *retrans* (retransmission) must be used with care.

The NFS protocol on DPS 7000 as a client performs five retransmissions (default) before giving up. Then it waits for 1.1 seconds (default) before trying again and so on. Other machines mounting OPEN 7 directories containing UFAS files must have these values. Otherwise, values for *timeo* can be greater than 1.1 seconds but not less. Values for *retrans* must be less than or equal to 4.

EXAMPLES

To mount a local disk:	`mount /dev/xy0g /usr`
To fake an entry for *nd* root:	`mount -ft sysV /dev/nd0 /`
To mount all *sysV* filesystems:	`mount -at sysV`
To mount an NFS remote filesystem:	`mount -t nfs serv:/usr/src /usr/src`
To mount an NFS remote filesystem:	`mount serv:/usr/src /usr/src`
To hard mount an NFS remote filesystem:	`mount -o hard serv:/usr/src /usr/src`
To save current mount state:	`mount -p > /etc/fstab`

FILES

`/etc/mtab`	table of mounted filesystems
`/etc/fstab`	table of filesystems mounted at boot

RESTRICTIONS

Mounting filesystems containing garbage crashes the system.

SEE ALSO

mountd(1M), nfsd(1M), fstab(4), mtab(4), umount(1M)

6. mountall(1M)

NAME

mountall - mounts file systems

SYNOPSIS

```
mountall
```

DESCRIPTION

The mountall command mounts file systems according to the file system table */etc/fstab*.

It is written to depend on as few commands as possible.

FILES

/etc/fstab

Format of /etc/fstab:

column 1	block special file name of file system (host:pathname form for nfs).
column 2	mount-point directory
column 3	file system type: sysV or nfs.
column 4	list of options, separated by commas, without spaces: ro|rw for read-only or read-write common options, and also nfs-specific options.
column 5+	ignored

Columns are separated by spaces or tabulations.

Lines beginning with "#" are comments. Empty lines are ignored.

SEE ALSO

umountall(1M)

6. mountd(1M)

NAME

mountd - NFS mount request server

SYNOPSIS

```
/etc/mountd [ -n ]
```

DESCRIPTION

mountd is an RPC server that answers filesystem mount requests. It reads the file */etc/xtab*, described in *exports*(4), to determine which filesystems are available for mounting by which machines. It also provides information as to which filesystems are mounted by which clients. This information can be printed using the *showmount*(1M) command.

OPTIONS

-n No check that clients are root users. Though this option decreases security, it does allow older versions (pre-3.0) of client NFS to work.

FILES

```
/etc/xtab
```

SEE ALSO

exports(4), showmount(1M)

6. mt(7)

NAME

mt - standard UNIX tape driver adapted to the DPS 7000

DESCRIPTION

The standard for tapes is:

```
/dev/{r}mt/#{hmlz}[n]
```

r	indicates a pure raw device. Data is read and written in blocks corresponding to the length specified in the read or write system call. By default, the blocks are read or written with a block size of 512 bytes.
mt	indicates a magnetic block device.
#	is the virtual unit number.
hmlz	indicates the density. h (high) specifies 6250 bpi, m (medium specifies 1600 bpi, l (low) specifies 800 bpi, z takes the density of the mounted tape.
n	indicates no rewind on CLOSE. The default is rewind.
v	indicates verbose mode (major 11).

Create device numbers

The program *mknod*(1M) creates special files in the */dev* directory as below:

```
mknod /dev/mt/<name> <major> <minor>
```

with 8-bit minor xwyyzzzz:

x = 0	with rewind
x = 1	without rewind
w = 0	512-byte block size
w = 1	raw mode
yy = 00	density 800 bpi
yy = 01	density 1600 bpi
yy = 10	density 6250 bpi
yy = 11	takes the density of the mounted tape.
zzzz	unit number.

with a byte major s:

s - 9	A mount message appears at the GCOS 7 operator console, then the tape can be mounted and named with the volume name asked by the OPEN 7 driver. The OPEN 7 driver blocks in the case where three tapes are waiting simultaneously for mount. After naming one tape the treatment starts.
s -11	The tape must be mounted and named before its use. No mount message is issued at the GCOS 7 console.

Naming volumes:

For GCOS 7, the OPEN 7 tapes (cpio or tar format) are considered as non standard tapes. The OPEN 7 driver asks for a volume name to be mounted in the GCOS 7 device.

The construction of the volume name is as follows:

```
UXTP# (with # as the virtual unit number )
```

At the GCOS 7 console, the command NV gives the correspondence between the volume name and the device where the tape is mounted. The syntax is as follows:

```
NV MTxx UXTP#
```

where xx is the GCOS 7 device number.

Configuration

Major device 9 mount message to operator if the tape is not mounted

Major device 11 the tape must be pre-mounted.

The number of minor entries is given by the define NTP in *sys/config.h*. The default value is 2 (#define NTP 2).

The maximum block size is configured to 10240 bytes in raw mode. This value can be changed by the define NTPSZ in *sys/config.h*. The value can be increased to 32767 bytes.

EXAMPLES

`/dev/rmt/0m` is created with the command:

```
mknod /dev/rmt/0m c 9 80
```

At the GCOS 7 console the mounted tape is named by:

```
NV MT01 UXTP0
```

(with the tape mounted on MT01 device)

6. mtab(4)

/etc/mtab is the mounted file systems table. See fstab(4), mount(1M), mountall(1M).

6. mv(1)

NAME

mv - move or rename files

SYNOPSIS

```
mv [-i] [-f] [-] file1 file2
mv [-i] [-f] [-] file ... directory
```

DESCRIPTION

mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod(2)*) and reads the standard input to obtain a line; if the line begins with y, the move takes place; if not, *mv* exits.

In the second form, one or more files (plain files or directories) are moved to the directory with their original file-names.

mv refuses to move a file onto itself.

Options:

-i	stands for interactive mode. Whenever a move is to supersede an existing file, the user is prompted by the name of the file followed by a question mark. If he answers with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.
-f	stands for force. This option overrides any mode restrictions or the -i switch.
-	means interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1), ln(1).

BUGS

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

6. mvdir(1M)

NAME

mvdir - moves a directory

SYNOPSIS

```
mvdir dirname name
```

DESCRIPTION

mvdir moves directories within a file system.

dirname must be a directory. If *name* does not exists, it will be created as a directory. If *name* exists, *dirname* will be created as *name/dirname*. *dirname* and *name* must not be on the same path; that is, one may not be subordinate to the other. For example, the command (1) is legal, but the command (2) is not:

```
mvdir x/y x/z       (1)
mvdir x/y x/y/z     (2)
```

SEE ALSO

mkdir(1), mv(1).

7. Commands Beginning with the Letter "N"

7. named(1M)

NAME

named - Internet domain name server

SYNOPSIS

```
named [-d debuglevel] [ -p port#] [bootfile]
```

DESCRIPTION

Named is the Internet domain name server (see RFC883 for more details). Without any arguments, named will read the default boot file */etc/named.boot* , read any initial data and listen for queries.

Options are:

-d debuglevel	Print debugging information. A number after the "d" determines the level of messages printed.
-p port#	Use a different port number. The default is the standard port number as listed in */etc/services*.

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. The following is a small example:

```
;
;   boot file for name server
;
; type      domain           source file or host
;
domain      berkeley.edu
primary     berkeley.edu     named.db
secondary   cc.berkeley.edu 10.2.0.78 128.32.0.10
cache       .                named.ca
```

The first line specifies that "berkeley.edu" is the domain for which the server is authoritative.

The second line states that the file "named.db" contains authoritative data for the domain "berkeley.edu". The file "named.db" contains data in the master file format described in RFC883 except that all domain names are relative to the origin ; in this case, "berkeley.edu" (see below for a more detailed description).

The second line specifies that all authoritative data under "cc.berkeley.edu" is to be transferred from the name server at 10.2.0.78. If the transfer fails it will try 128.32.0.10 and continue trying the address, up to 10, listed on this line.

The secondary copy is also authoritative for the specified domain.

The fourth line specifies data in "named.ca" is to be placed in the cache (i.e., well known data such as locations of root domain servers).

The file "named.ca" is in the same format as "named.db".

The master file consists of entries of the form:

```
$INCLUDE <filename>
$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_ttl* field is an optional integer number for the time-to-live field. It defaults to zero.

The *opt_class* field is the object address type; currently only one type is supported, for objects connected to the DARPA Internet.

The *type* field is 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 (5 numbers (see RFC883))
MB	a mailbox domain name (domain)
MG	a mail group member (domain)
MR	a mail rename domain name (domain)
MX	a mail exchange record
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)

NOTES

The following signals have the specified effect when sent to the server process using the *kill(1)* command.

SIGHUP	Causes server to read named.boot and reload database.
SIGINT	Dumps current data base and cache to /usr/tmp/named_dump.db
SIGUSR1	Turns on debugging; each SIGUSR1 increments debug level.
SIGUSR2	Turns off debugging completely.

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 servers database

SEE ALSO

kill(1), gethostbyname(3N), signal(2), resolver(3), resolver(4)

RFC882, RFC883, RFC973, RFC974

DPX/20 System Management Guide: Communications and Networks.

7. ncheck(1M)

NAME

ncheck - generates names from i-numbers (standard UNIX command adapted to OPEN 7)

SYNOPSIS

```
/etc/ncheck [ -i numbers ] [ -a ] [ -s ] [ filesystem ]
```

DESCRIPTION

The *ncheck* command generates names from i-numbers. *ncheck* is a standard UNIX command adapted to OPEN 7. It gives the literal name of a file from its i-node number and its file system. *ncheck* also indicates the GCOS 7 file name of the volume (= file system) on which the i-node is located.

ncheck with no argument generates a pathname and i-number list of all files on a set of default file systems. Names of directory files are followed by /..

The options are:

-i	reduces the report to only those files whose i-numbers follow.
-a	allows printing of the names . and .., which are ordinarily suppressed.
-s	reduces the report to special files and files with set-user-ID mode. This can be used to discover concealed violations of security policy.

filesystem must be specified by the system's *special* file.

The report is in no meaningful order, and may require sorting.

The name of the GCOS 7 file corresponding to *filesystem* is printed.

DIAGNOSTICS

When the filesystem structure is improper, ?? denotes the parent of a parentless file and a pathname beginning with ... denotes a loop.

SEE ALSO

fsck(1M), sort(1).

7. netgroup(4)

NAME

netgroup - list of network groups

DESCRIPTION

The *netgroup* file defines network wide groups, used for permission checking when doing remote mounts.

The information in *netgroup* is used to classify machines and users; each line of the *netgroup* file defines a group and has the format:

```
groupname member1 member2 ....
```

where *memberi* is either another group name, or a triple:

```
(hostname, username, domainname)
```

Any of three fields can be empty, in which case it signifies a wild card. Thus

```
universal (,,)
```

defines a group to which everyone belongs.

Field names that begin with something other than a letter, digit or underscore (such as "-") work in precisely the opposite fashion. For example, consider the following entries:

```
justmachines   (analytica,-,mts)
justpeople     (-,babbage,mts)
```

The machine *analytica* belongs to the group *justmachines* in the INET domain *mts*, (see *domainname(1)*) but no users belong to it.

Similarly, the user *babbage* belongs to the group *justpeople* in the domain *mts*, but no machines belong to it.

FILES

/etc/netgroup

SEE ALSO

getnetgrent(3N), exports(4)

7. netintro(7)

Referred to in ifconfig(1M)

7. netrc(4)

NAME

netrc - login file for remote networks

DESCRIPTION

If the *.netrc* file exists, it will be used by *ftp(1)* for automatic login on the remote host. For each remote host, the file contains a one-line entry that describes the login data for the user on that host.

An entry may consist of up to three blank-separated fields introduced by keywords. the keyword is followed by the literal data needed for login. The following keywords are available:

machine	the hostname of the machine.
login	the user login name for that host.
password	(Optional) the user's password on that host. **NOTE:** the literal password must be given in clear text; it is not encrypted.

If the *.netrc* file includes the password feature, permissions on the file must be set to prohibit reading by group and others; the file will not otherwise take effect.

EXAMPLE

The following example entry allows automatic login on the "admin" host by a user named "superuser" whose password is "open".

```
machine admin login superuser password open
```

FILES

$HOME/.netrc

SEE ALSO

ftp(1)

WARNING

For security reasons, use of the password feature is not recommended.

7. netstat(1)

NAME

netstat - show INET network status

SYNOPSIS

```
netstat [-AaimnrsSx] [-f address_family][-I interface]
        [-p protocol] [-a] [interval] [system] [corefile]
```

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meaning:

-A show the address of any associated protocol control blocks; used for debugging

-a show the state of all sockets; normally sockets used by server processes are not shown

-h show the state of the IMP host table

-f limit statistics and control block to *address_family*. The only *address_family* currently supported is **inet**.

-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)

-I show interface state for *interface* only.

-m show statistics recorded by the memory management routines (the network manages a "private share" of memory)

-n show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically)

-p " proto" show the state of sockets utilizing protocol *proto* ; the protocol is specified symbolically, and may be any protocol listed in the file */etc/protocols*.

-r show the routing tables

-s show per-protocol statistics

-S show serial line configuration

-x[ris] show X25 routing tables, interface status or interface statistics for *ix* interfaces only.

The arguments, *system* and *core* allow substitutes for the defaults **/unix** and **/dev/kmem**.

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, 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 in the Internet standard "dot format"; refer to *rhosts(4)* for more information regarding this format. 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 address (currently Internet specific) 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 the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. 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 then discard it. 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 an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. 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

iostat(1), vmstat(1), hosts(4), networks(4), protocols(4), services(4)

7. network(1M)

Referred to in ipx25 and xa(7).

7. network(7)

NAME

networking - introduction to networking facilities

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/route.h>
#include <net/if.h>
```

DESCRIPTION

This manual page briefly describes the networking facilities available in the system. Protocol-families , protocols , and "network interfaces" are described in *intro(7)*.

All network protocols are associated with a specific protocol-family .

Protocol Family
A protocol-family:

- provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport.
- may support multiple methods of addressing, though the current protocol implementations do not.
- is normally comprised of a number of protocols, one per *socket(2)* type. It is not required that a protocol-family support all socket types.
- may contain multiple protocols supporting the same socket abstraction.

Protocol
A protocol supports one of the socket abstractions detailed in *socket(2)*.

A specific protocol may be accessed either by creating a socket of the appropriate type and protocol-family, or by requesting the protocol explicitly when creating a socket.

Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol-family/network architecture.

Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

Network Interface
A network interface is similar to a device interface.

Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families, and/or address formats.

The SYNOPSIS section of each network interface entry gives a sample specification of the related drivers for use in providing a system description to the configuration phase.

The DIAGNOSTICS section lists messages which may appear on the console and in the system error log */usr/adm/syslog* due to errors in device operation.

PROTOCOLS

The system currently supports only the DARPA Internet protocols fully. Raw socket interfaces are provided to IP protocol layer of the DARPA Internet on top of Ethernet interfaces.

Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

ADDRESSING

Associated with each protocol family is an address for mat. The following address formats are used by the system:

```
#define   AF_UNIX   1   /* local to host (pipes) */
#define   AF_INET   2   /* internetwork: UDP, TCP, etc. */
```

ROUTING

The network facilities provided limited packet routing.

A simple set of data structures comprise a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host.

A user process, the routing daemon, maintains this data base with the aid of two socket specific *ioctl(2)* commands, SIOCADDRT and SIOCDELRT. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by super-user.

A routing table entry has the following form, as defined in *< net/route.h >*;

```
struct rtentry {
     u_long   rt_hash;
     struct   sockaddr rt_dst;
     struct   sockaddr rt_gateway;
     short    rt_flags;
     short    rt_refcnt;
     u_long   rt_use;
     struct   ifnet *rt_ifp;
};
```

with rt_flags defined from

```
#define   RTF_UP        0x1   /* route usable */
#define   RTF_GATEWAY   0x2   /* destination is a gateway */
#define   RTF_HOST      0x4   /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route).

When the system is booted, each network interface autoconfigured installs a routing table entry when it wishes to have packets sent through it. Normally the interface specifies the route through it is a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (i.e. the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (*rt_refcnt* is non-zero), the resources associated with it will not be reclaimed until further references to it are released.

The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existant entry, or ENOBUFS if insufficient resources were available to install a new route.

User processes read the routing tables through the */dev/kmem* device. The *rt_use* field contains the number of packets sent along the route.

This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, *lo(7)*, do not.

At boot time each interface which has underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address it is expected to install a routing table entry so that messages may be routed through it. Most interfaces require some part of their address specified with an SIOCSIFADDR ioctl before they will allow traffic to flow through them.

On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner.

On interfaces which provide dynamic network-link layer address mapping facilities (e.g. 10Mb/s Ethernets), the entire address specified in the ioctl is used. The following ioctl calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an ifrequest structure as its parameter.

This structure has the form:

```
struct   ifreq {
   char   ifr_name[16];   /* name of interface (e.g. "ec0") */
   union {
      struct   sockaddr ifru_addr;
      struct   sockaddr ifru_dstaddr;
      short    ifru_flags;
   } ifr_ifru;

#define        ifr_addr   ifr_ifru.ifru_addr   /* address */
#define        ifr_dstaddr   ifr_ifru.ifru_dstaddr
               /* other end of p-to-p link */
#define        ifr_flags   ifr_ifru.ifru_flags   /* flags */
};
```

SIOCSIFADDR	Set interface address. Following the address assignment, the "initialization" routine for the interface is called.
SIOCGIFADDR	Get interface address.
SIOCSIFDSTADDR	Set point to point address for interface.
SIOCGIFDSTADDR	Get point to point address for interface.
SIOCSIFFLAGS	Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.
SIOCGIFFLAGS	Get interface flags.
SIOCGIFCONF	Get interface configuration list. This request takes an *ifconf* structure (see below) as a value-result parameter. The *ifc_len* field should be initially set to the size of the buffer pointed to by *ifc_buf*. On return it will contain the length, in bytes, of the configuration list.

```
/*
* Structure used in SIOCGIFCONF request.
* Used to retrieve interface configuration
* for machine (useful for programs which
* must know all networks accessible).
*/

struct   ifconf {
         int       ifc_len;  /* size of associated buffer */
         union {

                   caddr_t   ifcu_buf;
                   struct    ifreq *ifcu_req;
         } ifc_ifcu;

#define  ifc_buf   ifc_ifcu.ifcu_buf   /* buffer address */
#define  ifc_req   ifc_ifcu.ifcu_req
                   /* array of structures returned */
};
```

SEE ALSO

ioctl(2), socket(2)

7. networks(4)

NAME

networks - names and numbers for the internet

SYNOPSIS

```
/etc/networks
```

DESCRIPTION

The file /etc/networks lists networks on the internet. Each line describes a single network and consists of the following blank separated fields:

```
name number aliases ...
```

where

name	is the official name of the network. All hosts on the internet should use the same official name for a given network.
number	is the network number, which serves as part of the DARPA Internet address for each host on the internet. All hosts on the internet must use the same number for a given network.
aliases ...	is a blank-separated list of local aliases for the network.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

EXAMPLE

```
# Building 1 Internet
Engineering 1   #R&D
Production 2    #Administration, etc.
```

FILES

/etc/networks

SEE ALSO

hosts(4).

7. newform(1)

NAME

newform - change the format of a text file

SYNOPSIS

```
newform [-s] [-itabspec] [-otabspec] [-bn] [-en] [-pn] [-an] [-f]
        [-cchar] [-ln] [files]
```

DESCRIPTION

newform reads lines from the named files, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for -s, command line options may appear in any order, may be repeated, and may be intermingled with the optional files. Command line options are processed in the order specified. This means that option sequences like "-e15 -l60" will yield results different from "-l60 -e15".

Options are applied to all files on the command line.

-s

Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.
For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e file-name
```

-itabspec

Input tab specification: expands tabs to spaces, according to the tab specifications given. Tabspec recognizes all tab specification forms described in *tabs(1)*. In addition, tabspec may be --, in which case *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec(4)*). If no tabspec is given, tabspec defaults to -8. A tabspec of -0 expects no tabs; if any are found, they are treated as -1.

-otabspec	Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for -itabspec. If no tabspec is given, tabspec defaults to -8. A tabspec of -0 means that no spaces will be converted to tabs on output.
-bn	Truncate n characters from the beginning of the line when the line length is greater than the effective line length (see -ln). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when -b with no n is used. This option can be used to delete the sequence numbers from a COBOL program as follows: `newform -l1 -b7 file-name`
-en	Same as -bn except that characters are truncated from the end of the line.
-pn	Prefix n characters (see -ck) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.
-an	Same as -pn except characters are appended to the end of a line.
-f	Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the last -o option. If no -o option is specified, the line which is printed will contain the default specification of -8.
-ck	Change the prefix/append character to k. Default character for k is a space.
-ln	Set the effective line length to n characters. If n is not entered, -l defaults to 72. The default line length without the -l option is 80 characters. Note that tabs and backspaces are considered to be one character (use -i to expand tabs to spaces). The -l1 must be used to set the effective line length shorter than any existing line in the file so that the -b option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not -s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.

tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspe*c referencing another file (or standard input).
0	normal execution
1	for any error

SEE ALSO

csplit(1), tabs(1).

fspec(4) in the *Programmer's Reference Manual*.

BUGS

newform normally only keeps track of physical characters; however, for the -i and -o options, newform will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of -i-- or -o--).

If the -f option is used, and the last -o option specified was -o--, and was preceded by either a -o-- or a -i--, the tab specification format line will be incorrect.

7. newgrp(1M)

NAME

newgrp - log in to a new group

SYNOPSIS

```
newgrp [-] [group]
```

DESCRIPTION

The *newgrp* command changes a user's group identification.

The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null.

System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values.

For example, a user has a primary prompt string (PSI) other than $ (default and has not exported PSI. After an invocation of *newgrp*, successful or not, their PSI will now be set to the default prompt string $.

Note that the shell command export (see *sh(1)*) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a -, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in */etc/group* as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5)

BUGS

There is no convenient way to enter a password into */etc/group*. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

7. nfsd(1M)

NAME

nfsd, biod - NFS daemons

SYNOPSIS

```
/etc/nfsd [nservers]
/etc/biod [nservers]
```

DESCRIPTION

nfsd starts the NFS server daemons that handle client filesystem requests.

nservers is the number of file system request daemons to start. This number should be based on the load expected on this server.

biod, run on an NFS client, starts *nservers* asynchronous block I/O daemons, which do read-ahead and write-behind of blocks from the client's buffer cache.

SEE ALSO

mountd(1M), exports(4)

7. nfsstat(1M)

NAME

nfsstat - Network File System statistics

SYNOPSIS

```
nfsstat [-csnrz]
```

DESCRIPTION

The *nfsstat* command displays statistical information about the Network File System (NFS) and Remote Procedure Call (RPC) interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is *nfsstat -csnr*, that is, print everything and reinitialize nothing.

OPTIONS

-c	Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **-n** and **-r** options to print client NFS or client RPC information only.
-s	Display server information. Works like the **-c** option above.
-n	Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **-c** and **-s** options to print client or server NFS information only.
-r	Display RPC information. Works like the **-n** option above.
-z	Zero (reinitialize) statistics. Can be combined with any of the above options to zero particular sets of statistics after printing them. The user must have write permission on */dev/kmem* for this option to work.

FILES

/unix	system namelist
/dev/kmem	kernel memory

7. nice(1)

NAME

nice - run a command at low priority

SYNOPSIS

```
nice [-increment] command [arguments]
```

DESCRIPTION

nice executes command with a lower CPU scheduling priority. If the increment argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., --10.

SEE ALSO

nohup(1), nice(2)

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An increment larger than 19 is equivalent to 19.

7. nice(2)

Subroutine referred to in gnice(1M) and nice(1); See the *Programmer's Reference Manual*.

7. nl(1)

NAME

nl - number lines in a file

SYNOPSIS

```
nl [-htype] [-btype] [-ftype] [-vstart#] [-iincr] [-p] [-lnum]
   [-ssep] [-wwidth] [-nformat] [-ddelim] file
```

DESCRIPTION

The *nl* command reads lines from the named file or the standard input if no file is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	Start of
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

-btype — Specifies which logical page body lines are to be numbered. Recognized types and their meaning are:

a	number all lines
t	number lines with printable text only
n	no line numbering
pstring	number only lines that contain the regular expression specified in string.

Default type for logical page body is t (text lines numbered).

-htype — Same as -btype except for header. Default type for logical page header is n (no lines numbered).

-ftype	Same as -btype except for footer. Default for logical page footer is n (no lines numbered).
-vstart#	Start# is the initial value used to number logical page lines. Default is 1.
-iincr	Incr is the increment value used to number logical page lines. Default is 1.
-p	Do not restart numbering at logical page delimiters.
-lnum	Num is the number of blank lines to be considered as one. For example, -l2 results in only the second adjacent blank being numbered (if the appropriate -ha, -ba, and/or -fa option is set). Default is 1.
-ssep	Sep is the character(s) used in separating the line number and the corresponding text line. Default sep is a tab.
-wwidth	Width is the number of characters to be used for the line number. Default width is 6.
-nformat	Format is the line numbering format. Recognized values are: ln, left justified, leading zeroes suppressed; rn, right justified, leading zeroes supressed; rz, right justified, leading zeroes kept. Default format is rn (right justified).
-dxx	The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the -d and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The following command will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

```
nl -v10 -i10 -d!+ file1
```

SEE ALSO

pr(1)

7. nlist(3C)

Referred to in sysdef(1M). See the *Programmer's Reference Manual*.

7. nm(1P)

NAME

nm - print name list of common object file

SYNOPSIS

```
nm [-oxhvnefurpVT] filename ...
```

DESCRIPTION

The *nm* command displays the symbol table of each common object file, *filename*. *filename* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name	The name of the symbol.
Value	Its value expressed as an offset or an address depending on its storage class.
Class	Its storage class.
Type	Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., char[n][m]). Note that the object file must have been compiled with the -g option of the *cc(1)* command for this information to appear.
Size	Its size in bytes, if available. Note that the object file must have been compiled with the -g option of the *cc(1)* command for this information to appear.
Line	The source line number at which it is defined, if available. Note that the object file must have been compiled with the -g option of the *cc(1)* command for this information to appear.
Section	For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of *nm* may be controlled using the following options:

-o	Print the value and size of a symbol in octal instead of decimal.
-x	Print the value and size of a symbol in hexadecimal instead of decimal.
-h	Do not display the output header data.

-v	Sort external symbols by value before they are printed.
-n	Sort external symbols by name before they are printed.
-e	Print only external and static symbols.
-f	Produce full output. Print redundant symbols (.text, .data, .lib, and .bss), normally suppressed.
-u	Print undefined symbols only.
-r	Prepend the name of the object file or archive to each output line.
-p	Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters: U (undefined), A (absolute), T (text segment symbol), D (data segment symbol), S (user defined segment symbol), R (register symbol), F (file symbol), C (common symbol). If the symbol is local (non-external), the type letter is in lower case.
-V	Print the version of the nm command executing on the standard error output.
-T	By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The -T option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, the two following commands print the static and external symbols in *name*, with external symbols sorted by value:

```
nm name -e -v
nm -ve name
```

RETURNS

"nm: name: cannot open" if name cannot be read.

"nm: name: bad magic" if name is not a common object file.

"nm: name: no symbols" if the symbols have been stripped from name.

APPLICATION USAGE

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the -v and -n options should be used only in conjunction with the -e option.

FILES

TMPDIR/*	temporary files. TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable TMPDIR (see *tempnam(3S)*).

SEE ALSO

as(1), cc(1), ld(1), tempnam(3S), a.out(4), ar(4).

7. nohup(1)

NAME

nohup - run a command immune to hangups and quits

SYNOPSIS

```
nohup command [arguments]
```

DESCRIPTION

nohup executes command with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to *nohup.out*. If nohup.out is not writable in the current directory, output is redirected to *$HOME/nohup.out*.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the nohup applies to everything in *file*. If the shell procedure file is to be executed often, then the need to type *sh* can be eliminated by giving file execute permission. Add an ampersand and the contents of file are run in the background with interrupts also ignored (see *sh(1)*):

```
nohup file &
```

An example of what the contents of file could be is:

```
sort ofile > nfile
```

FILES

nohup.out

SEE ALSO

chmod(1), nice(1), sh(1),

signal(2) in the *Programmer's Reference Manual*.

WARNINGS

```
nohup command1; command2                          (1)
nohup (command1; command2)                        (2)
```

(1) In this command, *nohup* applies only to command1.

(2) This command is syntactically incorrect.

8. Commands Beginning with the Letter "O"

8. od(1)

NAME

od - dumps files in octal, ASCII, and other formats

SYNOPSIS

```
od [-bcdosex] [file] [ [+]offset[.][b] ]
```

DESCRIPTION

od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, -o is default. The meanings of the format options are:

-b	Interpret bytes in octal.
-c	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
-d	Interpret words in unsigned decimal.
-ex	Interpret bytes in EBCDIC.
-o	Interpret words in octal.
-s	Interpret 16-bit words in signed decimal.
-x	Interpret words in hex.

The file argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If dot (.) is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in bytes per block according to the environment standard. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.

8. on(1)

NAME

on - execute a command remotely

SYNOPSIS

```
on [-i] [-n] [-d] host command [argument]...
```

DESCRIPTION

The *on* program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

Standard input is connected to standard input of the remote command, and standard output and standard error from the remote command are sent to the corresponding files for the *on* command.

OPTIONS

-i Interactive mode: use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

-n No Input: this option causes the remote program to get end-of-file when it reads from standard input, instead of passing standard input from the standard input of the *on* program. For example, **-n** is necessary when running commands in the background with job control.

-d Debug mode: print out some messages as work is being done.

SEE ALSO

rexd(1M), exports(4)

DIAGNOSTICS

unknown host	Host name not found
cannot connect to server	Host down or not running the server
can't find.	Problem finding the working directory
can't locate mount point	Problem finding current file system

Other error messages may be passed back from the server.

8. open7adm

A command used by the super administrator of OPEN 7. It displays menus that ease the management of the Internet network, the OPEN 7 system and the user access rights.

Its use is described in the *OPEN 7 Administrator's Guide*.

9. Commands Beginning with the Letter "P"

9. pack(1)

NAME

pack, pcat, unpack - compress and expand files

SYNOPSIS

```
pack [-] [-f] filename...

pcat filename...

unpack filename...
```

DESCRIPTION

pack Command

pack attempts to store the specified files in a packed form using Huffman (minimum redundancy) codes on a byte-by-byte basis. Wherever possible (and useful), each input file *filename* is replaced by a packed file *filename.z* with the same access modes, access and modified dates, and owner as those of filename. If *pack* is successful, *filename* will be removed.

Packed files can be restored to their original form using *unpack* or *pcat*.

The amount of compression obtained depends on the size of the input file and the frequency distribution of its characters.

Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks unless the distribution of characters is very skewed. This may occur with printer plots or pictures.

Typically, large text-files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression. Their packed versions come in at about 90% of the original size.

No packing will occur if:

- the file appears to be already packed
- the file name has more than 12 characters
- the file has links
- the file is a directory
- the file cannot be opened
- no disk storage blocks will be saved by packing
- a file called name.z already exists
- the .z file cannot be created
- an I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended .z extension.

Directories cannot be packed.

pcat Command

pcat does for packed files what *cat(1V)* does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output.

```
pcat filename.z                          }  (1)
pcat filename                            }

pcat filename> newname                      (2)
```

(1) To view a packed file named *name.z*, use one of these commands.

(2) To make an unpacked copy without destroying the packed version, use this command.

Failure may occur if:

- the filename (exclusive of the .z) has more than 12 characters;
- the file cannot be opened;
- the file does not appear to be the output of pack.

unpack Command

unpack expands files created by *pack*. For each file filename specified in the command, a search is made for a file called *filename.z* (or just *filename*, if *filename* ends in .z). If this file appears to be a packed, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

- a file with the "unpacked" name already exists,
- the unpacked file cannot be created.

OPTIONS

-	Print compression statistics for the following filename or names on the standard output. Subsequent '-'s between filenames toggle statistics off and on.
-f	Force packing of *filename*. This is useful for causing an entire directory to be packed, even if some of the files will not benefit.

DIAGNOSTICS

pack returns the number of files that it failed to compress.

pcat returns the number of files it was unable to unpack.

unpack returns the number of files it was unable to unpack.

SEE ALSO

cat(1)

9. passwd(1)

NAME

passwd - change login password

SYNOPSIS

```
passwd [name]
```

DESCRIPTION

This command is a security administration utility. It changes or installs a password associated with the login name.

Ordinary users may change only the password which corresponds to their login name.

The *passwd* command prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and *passwd* terminates; see *passwd(4)*.

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower case letters.
- Each password must differ from the user's login name and any reverse or circular shift of that login name.

For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent. New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

One user whose effective user ID is zero is called a super-user; see *id(1)*, and *su(1M)*. Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

NOTES

C2 Secure Environment
In a Secure environment, this command can be used only by the user with authentication subsystem privilege.

The *passwd* command has been moved to the */tcb/bin* directory.

SEE ALSO

login(1), passwd(4), crypt(3C), id(1M), su(1M)

9. passwd(4)

NAME

passwd - password file

DESCRIPTION

The passwd file is an ASCII file which contains for each user the following information:

- login name
- encrypted password
- numerical user ID
- numerical group ID
- gecos information or general information
- initial working directory
- program to use as shell

Each field within each user's entry is separated from the next by a colon.

The gecos field is used only when communicating with that system (gecos job number, box number, optional gecos user ID), and in other installations can contain any desired information.

Each user is separated from the next by a new-line.

If the password field is null, no password is demanded; if the shell field is null, the shell itself is used.

This file resides in directory */etc*. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user IDs to names.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (., /, 0-9, A-Z, a-z), except when the password is null, in which case the encrypted password is also null. Password aging is effected for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.)

The first character of the age, M say, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to supply a new one.

The next character, m say, denotes the minimum period in weeks which must expire before the password may be changed.

The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) M and m have numerical values in the range 0-63 that correspond to the 64-character alphabet shown above (i.e., / = 1 week; z = 63 weeks).

If m = M = 0 (derived from the string . or ..) the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If m > M (signified, e.g., by the string ./) only the super-user will be able to change the password.

FILES

/etc/passwd

SEE ALSO

a64l(3C), getpwent(3C), group(4), login(1), passwd(1).

9. paste(1)

NAME

paste - join corresponding lines of several files, or subsequent lines of one file

SYNOPSIS

```
paste filename1 filename2 ...

paste -dlist filename1 filename2 ...

paste -s [-dlist] filename
```

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *filename1*, *filename2*, etc.

It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of *cat(1)* which concatenates vertically, that is, one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the TAB character, or with characters from an optionally specified list. *paste* can be used as a filter, if - is used in place of a filename.

OPTIONS

-d

Without this option, the NEWLINE characters of each but the last file (or last line in case of the -s option) are replaced by a TAB character. This option allows replacing the TAB character by one or more alternate characters in list. The list is used circularly; when exhausted, it is reused. In parallel merging (no -s option), the lines from the last file are always terminated with a NEWLINE character, not from the list.

list may contain the special escape sequences: \n (NEW-LINE), \t (tab), \\ (backslash), and \0 (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell.

-s

Merge subsequent lines rather than one from each input file. Use TAB for concatenation, unless *list* is specified with -d. Regardless of the list, the very last character of the file is forced to be a NEWLINE.

EXAMPLES

```
ls | paste----                                    (1)
paste -s -d"\t\n" filename                        (2)
```

(1) List directory in four columns.

(2) Combine pairs of lines into lines.

SEE ALSO

cat(1V), cut(1V), grep(1V), pr(1V)

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for -s option, no more than 12 input files may be specified.

9. pcat(1)

Expands files packed with pack(1). See pack(1).

9. pg(1)

NAME

pg - page through a file on a soft-copy terminal

SYNOPSIS

```
/usr/bin/pg [-cefns]  [-number]  [-p string]  [+linenumber]
            [+/pattern/] [filename ...]
```

DESCRIPTION

pg is a filter that allows you to page through *filename*, one screenful at a time, on a soft-copy terminal. With a filename of '-', or no filename specified, *pg* reads from the standard input. Each screenful is followed by a prompt. If the user types a RETURN, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo(5V)* data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type *dumb* is assumed.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding address, an optionally signed number indicating the point from which further text should be displayed. This address is interpreted in either pages or lines depending on the command. A signed address specifies a point relative to the current page or line, and an unsigned address specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1) NEWLINE or SPACE	Display one page. The address is specified in pages.
(+1) l	With a relative address *pg* will simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.
(+1) d or ^D	Simulate scrolling half a screen forward or backward.

The following perusal commands take no address.

. or ^L	Redisplay the current page of text.
$	Display the last full window in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed(1)* are available. They must always be terminated by a NEWLINE, even if the -n option is specified.

i/pattern/	Search forward for the ith (default i=1) occurrence of pattern. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.
i^pattern^ i?pattern?	Search backwards for the ith (default i=1) occurrence of pattern. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for ADDS 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending m or b to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix t can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

in	Begin perusing the ith next file in the command line. The i is an unsigned number, default value is 1.
ip	Begin perusing the ith previous file in the command line. i is an unsigned number, default value is 1.
iw	Display another window of text. If i is present, set the window size to i.
s filename	Save the input in the named file. Only the current file being perused is saved. The white space between the s and filename is optional. This command must always be terminated by a NEWLINE, even if the -n option is specified.
h	Help by displaying an abbreviated summary of available commands.
q or Q	Quit *pg*.
!command	command is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a NEWLINE, even if the -n option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key, normally CTRL-\ or the BREAK (interrupt) key. This causes *pg* to stop sending output, and display the 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 the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat(1V)*, except that a header is printed before each file (if there is more than one).

OPTIONS

The command line options are:

-number	An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).
-p string	Use string as the prompt. If the prompt string contains a '%d', the first occurrence of '%d' in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ':'.
-c	Home the cursor and clear the screen before displaying each page. This option is ignored if *clear_screen* is not defined for this terminal type in the *terminfo(5V)* data base.
-e	Do not pause at the end of each file.
-f	Inhibit *pg* from splitting lines. Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (for instance, escape sequences for underlining) generate undesirable results.
-n	Automatic end of command as soon as a command letter is entered. Normally, commands must be terminated by a NEWLINE character.
-s	Print all messages and prompts in standout mode (usually inverse video).
+linenumber	Start up at *linenumber*.
+/pattern/	Start up at the first line containing the regular expression pattern.

EXAMPLES

A sample usage of pg in reading system news would be

```
news | pg -p '(Page %d):'
```

FILES

/usr/share/lib/terminfo/*	terminal information data base
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

cat(1), crypt(1), ed(1), grep(1), more(1), terminfo(4)

BUGS

If terminal TAB characters are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options (for instance, *crypt(1)*), terminal settings may not be restored correctly.

NOTES

While waiting for terminal input, *pg* responds to BREAK, DEL, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of *more(1)* will find that the z and f commands are available, and that the terminal '/', '^', or '?' may be omitted from the searching commands.

9. ping(1M)

NAME

ping - send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

```
/etc/ping [-r] [-v] host [packetsize] [count]
```

DESCRIPTION

The *ping* command is a troubleshooting tool for tracking a single-point hardware or software failure in the Internet. It uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct timeval**, and then an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option.

Other options are:

-r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it.(for example, after the interface was dropped by *routed(1M)*).

-v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged".*ping* sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned.

No output is produced if there is no response. If an optional *count* is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

SEE ALSO

netstat(1), ifconfig(1M), traceroute(1M).

9. portmap(1M)

NAME

portmap - DARPA port to RPC program number mapper

SYNOPSIS

```
/etc/portmap
```

DESCRIPTION

portmap is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell *portmap* what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact *portmap* on the server machine to determine the port number where RPC packets should be sent.

SEE ALSO

rpcinfo(1M)

BUGS

If *portmap* crashes, all servers must be restarted.

9. ports(7)

Referred to in tty(7).

9. pr(1)

NAME

pr - prepare file(s) for printing, perhaps in multiple columns

SYNOPSIS (BSD)

```
pr [-|+n] [-fmt] [-hstring] [-ln] [-sc] [-wn] [filename]
```

SYNOPSIS (SYSTEM V)

```
/bin/pr [-|+n] [-adfmprt] [-eck] [-h string] [-ick] [-ln] [-nck]
        [-ok] [-sc] [-wn] [filename]
```

DESCRIPTION

The *pr* command prepares one or more filenames for printing. By default, the output is separated into pages headed by a date, the name of the file, and the page number. pr prints its standard input if there are no filename arguments. FORMFEED characters in the input files cause page breaks in the output, as expected.

By default, columns are of equal width, separated by at least one SPACE. Lines that do not fit are truncated. If the -s option is used, lines are not truncated and columns are separated by the separation character.

Inter-terminal messages using *write(1)* are forbidden during a pr.

OPTIONS

Options apply to all following filenames but may be reset between filenames:

-f — Use FORMFEED characters instead of NEWLINE characters to separate pages. A FORMFEED is assumed to use up two blank lines at the top of a page. Thus this option does not affect the effective page length.

-m — Print all filenames simultaneously, each in one column, for example:

```
Print       Print       The
the         the         third
lines       lines       file's
of          of          lines
file        file        go
one.        two.        here.
```

-t — Do not print the 5-line header or the 5-line trailer normally supplied for each page. Pages are not separated when this option is used, even if the -f option was used. The -t option is intended for applications where the results should be directed to a file for further processing.

-h string — Use string, instead of the file name, in the page header.

-ln — Take the length of the page to be n lines instead of the default value of 66.

-sc — Separate columns by the single character c instead of by the appropriate amount of white space. A missing c is taken to be a TAB.

-wn — For multicolumn output, take the width of the page to be n characters instead of the default value of 72.

-n — Produce n-column output. For example:

```
Print          of             in
the            one            three
lines          file           columns
```

Columns are not balanced. If, for example, there are as many lines in the file as there are lines on the page, only one column will be printed. Even if the -t option (see below) is specified, blank lines are printed at the end of the output to pad it to a full page.

+n — Begin printing with page n.

SYSTEM V OPTIONS

When the -n option is specified for multicolumn output, columns are balanced. For example, if there are as many lines in the file as there are lines to be printed, and two columns are to be printed, each column will contain half the lines of the file. If the -t option is specified, no blank lines are printed to pad the last page.

The options -e and -i are assumed for multicolumn output.

The -m option overrides the -k and -a options.

The -f option does not assume that FORMFEED uses up two blank lines; blank lines are printed after the FORMFEED if necessary. If the standard output is a terminal, -f will cause pr to wait for a RETURN before printing the first page.

-a — When combined with the n option, print multicolumn output across the page. For example: Print the lines of one file in three columns.

-d — Double-space the output.

-p — Pause before beginning each page if the output is directed to a terminal (pr rings the bell at the terminal and wait for a RETURN).

-r	Do not print diagnostic reports if a file cannot be opened, or if it is empty.
-eck	Expand input TAB characters to character positions k+1, 2*k+1, 3*k+1, etc. If k is 0 or is omitted, default TAB settings at every eighth position are assumed. TAB characters in the input are expanded into the appropriate number of SPACE characters. If c (any non-digit character) is given, it is treated as the input TAB character (default for c is the TAB character).
-ick	In output, replace white space wherever possible by inserting TAB characters to character positions k+1, 2*k+1, 3*k+1, etc. If k is 0 or is omitted, default TAB settings at every eighth position are assumed. If c (any non-digit character) is given, it is treated as the output TAB (default for c is the TAB character).
-nck	Provide k-digit line numbering (default for k is 5). The number occupies the first k+1 character positions of each column of normal output or each line of -m output. If c (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for c is a TAB).
-ok	Offset each line by k character positions. The number of character positions per line is the sum of the width and offset.

EXAMPLES

Print a file called dreadnought on the printer - this is the simplest use of pr:

```
example% pr dreadnought | lpr
example%
```

Produce three laminations of a file called ridings side by side in the output, with no headers or trailers, the results to appear in the file called Yorkshire:

```
example% pr -m -t ridings ridings ridings > Yorkshire
example%
```

FILES

`/dev/tty*`	to suspend messages.

SEE ALSO

cat(1), lpr(1)

DIAGNOSTICS

can't print 0 cols, using 1
-0 was specified as a -n option.

pr: bad key
An illegal option was given.

pr: No room for columns.
The number of columns requested do not fit on the page.

pr: Too many args
More than 10 files were specified with the -m option.

filename error: error
filename could not be opened. This diagnostic is not printed if pr is printing on a terminal.

SYSTEM V DIAGNOSTICS

pr: bad option
An illegal option was given.

pr: width too small
The number of columns requested does not fit on the page.

pr: too many files
More than 10 files were specified with the -m option.

pr: page-buffer overflow
The formatting required is more complicated than pr can handle.

pr: out of space
pr could not allocate a buffer that it required.

pr: filename -- empty file
filename was empty. This diagnostic is printed after all the files are printed if pr is printing to a terminal.

pr: can't open filename
filename could not be opened. This diagnostic is printed after all the files are printed if pr is printing to a terminal.

BUGS

The options described above interact with each other in strange and as yet undefined ways.

9. printcap(5)

NAME

printcap - printer capability database

SYNOPSIS

`/etc/printcap`

DESCRIPTION

The *printcap* database is a simplified version of the *termcap(5)* database for describing printers. The spooling system accesses the *printcap* file every time it is used, allowing dynamic addition and deletion of printers.

Each entry in the database describes one printer. This data base may not be substituted for, as is possible for *termcap*, because it may allow accounting to be bypassed.

The default printer is normally *lp*, though the environment variable PRINTER may be used to override this.

Each spooling utility supports a -Pprinter option to explicitly name a destination printer.

Each entry in the printcap file describes a printer, and is a line consisting of a number of fields separated by ':' characters.

The first entry for each printer gives the names which are known for the printer, separated by `|' characters. The first name is conventionally a number. The second name given is the most common abbreviation for the printer, and the last name given should be a long name fully identifying the printer. The second name should contain no blanks; the last name can contain blanks for readability.

Entries may continue onto multiple lines by giving a `\' as the last character of a line, and empty fields may be included for readability.

Capabilities in printcap are all introduced by two-character codes, and are of three types:

1) Boolean

Capabilities that indicate that the printer has some particular feature.

Boolean capabilities are simply written between the ':' characters, and are indicated by the word 'bool' in the type column of the capabilities table below.

2) Numeric

Capabilities that supply information such as baud-rates, number of lines per page, and so on.

Numeric capabilities are indicated by the word num in the type column of the capabilities table below. Numeric capabilities are given by the two-character capability code followed by the '#' character, followed by the numeric value. The following example is a numeric entry stating that this printer should run at 1200 baud:

```
:br#1200:
```

3) String

Capabilities that give a sequence which can be used to perform particular printer operations such as cursor motion.

String valued capabilities are indicated by the word "str" in the type column of the capabilities table below. String valued capabilities are given by the two-character capability code followed by a '=' sign and then a string ending at the following ':'.

For example:

```
:rp=spinwriter:
```

is a sample entry stating that the remote printer is named spinwriter.

CAPABILITIES

NameType	Default	Description
af	str NULL	name of accounting file
br	num	none if lp is a tty, set the baud rate (ioctl call)
df	str NULL	TeX data filter (DVI format)
du	str 0	User ID of user `daemon'.
fc	num 0	if lp is a tty, clear flag bits
ff	str "lf"	string to send for a form feed
o	bool false	print a form feed when device is opened
f	fs num 0	like 'fc' but set bits
hl	bool false	print the burst header page last
ic	bool false	driver supports (non standard) ioctl to indent printout
if	str NULL	name of input/communication filter (created per job)
lf	str "ldev/console"	error logging file name
lo	str "lock"	name of lock file
lp	str "/dev/lp"	device name to open for output
mc	num 0	maximum number of copies
ms	str NULL	list of terminal modes to set or clear
mx	num 1000	maximum file size(in BUFSIZ blocks), zero=unlimited
nd	str NULL	next directory for list of queues (unimplemented)
of	str NULL	name of output/banner filter (created once)
pc	num 200	price per foot or page in hundredths of cents
pl	num66	page length (in lines)
pw	num 132	page width (in characters)
px	num 0	page width in pixels (horizontal)
py	num 0	page length in pixels (vertical)
rg	str NULL	restricted group.Only members of group allowed access
rm	str NULL	machine name for remote printer
rp	str "lp"	remote printer name argument
rs	bool false	restrict remote users to those with local accounts
rw	bool false	open printer device read /write instead of write-only
sb	bool false	short banner (one line only)
sc	bool false	suppress multiple copies
sd	str "/var/spool/lpd"	spool directory
sf	bool false	suppress form feeds
sh	bool false	suppress printing of burst page header
st	str "status"	status file name
tc	str NULL	name of similar printer must be last
tr	str NULL	trailer string to print when queue empties
xc	num 0	if lp is a tty, clear local mode bits
xs	num 0	like `xc' but set bits

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

FILES

/etc/printcap

SEE ALSO

lpq(1), lpr(1), lprm(1), stty(1), termcap(5), lpc(1M), lpd(1M)

9. printers(8)

List of supported printers, used by lpadmin(1M)

9. profile(4)

The /etc/profile file contains the system's profile. Referred to in ksh(1) and su(1M). See the *Programmer's Reference Manual*.

9. profiler(1M)

NAME

profiler includes the prfld, prfstat, prfdc, prfsnap and prfpr programs to study the activity of the operating system.

SYNOPSIS

```
prfld [ system_namelist ]

prfstat on
prfstat off

prfdc file [ period [ off_hour ]]

prfsnap file

prfpr file [ cutoff [system_namelist]]
```

DESCRIPTION

prfld, prfstat, prfdc, prfsnap, and prfpr form a system of programs to facilitate an activity study of the operating system.

prfld is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from system_namelist.

prfstat is used to enable or disable the sampling mechanism. Profiler overhead is less than 1 % as calculated for 500 text addresses. *prfstat* will also reveal the number of text addresses being measured.

prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *prfdc* will store the counters into file every period minutes and will turn off at off_hour (valid values for off_hour are 0-24). *prfsnap* collects data at the time of invocation only, appending the counter values to file.

prfpr formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in system_namelist) and is printed if the percent activity for that range is greater than cutoff.

FILES

/dev/prf	interface to profile data and text addresses
/unix	default for system namelist file

9. protocols(4)

NAME

protocols - list of Internet protocols

DESCRIPTION

The file */etc/protocols* lists known DARPA Internet protocols. Each line describes a single protocol and consists of the following blank separated fields:

```
name number aliases ...
```

where

name	is the official name of the protocol.
number	is the protocol number.
aliases ...	is a blank-separated list of local aliases for the protocol.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Protocol names and numbers are specified by the DDN Network Information Center.

Do not change this file.

FILES

/etc/protocols

SEE ALSO

socket(2), slink(1M), ldsocket(1M)

9. ps(1)

NAME

ps - report process status

SYNOPSIS

```
ps [options]
```

DESCRIPTION

The *ps* command prints information about active processes. Without options, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name.

Otherwise, the information that is displayed is controlled by the selection of options.

The options are:

-e	Print information about all process.
-d	Print information about all processes, except process group leaders.
-a	Print information about all processes, except process group leaders and processes not associated with a terminal.
-f	Generate a full listing. (See below for meaning of columns in a full listing.)
-l	Generate a long listing. See below.
-t termlist	Restrict listing to data about the processes associated with the terminals given in termlist.

Terminal identifiers may be specified in one of two forms: the device's file name (e.g., ttyc3d1) or if the device's file name starts with tty, just the digit identifier (e.g., c3d1).

-p proclist	Restrict listing to data about processes whose process ID numbers are given in *proclist*.
-u uidlist	Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the -f option is used, in which case the login name will be printed.
-g grplist	Restrict listing to data about processes whose process group leaders are given in *grplist*.

Column Headings in a ps Listing

The column headings and the meaning of the columns in a *ps* listing are given below.

```
F  S  UID  PID  PPID   C   PRI   NI   SZ   WCHAN   STIME   TTY   TIME   CMD
```

In the description of the headings below:

- the letters **f** and **l** indicate the option (full or long) that causes the corresponding heading to appear;
- **all** means that the heading always appears.

Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F (l)
Flags (octal and additive) associated with the process:

0 swapped;
1 in core;
2 system process;
4 locked-in core (e.g., for physical I/O);
8 process cannot be awakened by a signal;
10 being swapped;
20 bring traced by another process;
40 another tracing flag;
80 process in stream poll.

S (l)
Indicates the state of the process:

0 non-existent;
S sleeping;
R running;
I intermediate;
Z terminated;
T stopped;
X owner cpu;
B swapped
M transfer execution from one cpu to the other.

UID (f,l)
The user ID number of the process owner; the login name is printed under the -f option.

PID (all)
The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l)
The process ID of the parent process.

C (f,l)
Processor utilization for scheduling.

PRI (l)
The priority of the process; higher numbers mean lower priority.

NI (l)
Nice value; used in priority computation.

SZ (l)
The size in blocks of the core image of the process.

WCHAN (l)
The event for which the process is waiting or sleeping; if blank, the process is running.

STIME (f)
Starting time of the process.

TTY (all)
The controlling terminal for the process.

TIME (all)
The cumulative execution time for the process.

CMD (all)
The command name; the full command name and its arguments are printed under the -f option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <defunct>.

Under the -f option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the -f option, is printed.

FILES

/dev/ps	memory
/dev	searched to find terminal ("tty") names

NOTES

C2 Secure Environment
The *ps* command list processes owned by login user ID, or owned by real user ID of current process on current terminal.

This command can be used only by the superuser with memory subsystem privilege.

SEE ALSO

acctcom(1), kill(1), nice(1)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

9. ptq(7)

NAME

ptq - passthrough quick device, an OPEN 7 special driver used to reach any DSA mailbox

DESCRIPTION

Passthrough quick device. *ptq* is an OPEN 7 special driver which is used to reach any DSA mailbox in the DSA network. This mailbox may be IOF, TDS, or OPEN 7 on the same DPS 7000 or on another DPS 7000 in the DSA network.

The passthrough quick device *ptq* provides the interface with the GCOS 7 VCAM module, to enable terminal oriented DSA sessions with a mailbox (IOF, TDS, or UNIX type) on a local or remote site.

This driver offers the following particularities:

- 2048 DSA sessions in its maximum configuration,
- real raw-mode data tranfer between user data-space and VCAM (no space reserved in kernel),
- Only raw mode is available with *ptq*, that is each read/write system call corresponds to a read/write at the VCAM interface,
- *ioctl*(2) command PTOPENCNX, used for direct opening of a DSA session through VCAM, without processing a *read*(2) or *write*(2) system call
- *ioctl*(2) command PTSENDTLG which is used to send telegrams through VCAM
- extension of *select*(2) system call to exceptional condition pending; three state flags describe the exceptional conditions: PTOPENACK, PTDMDTURN, and PTABNTERM
- non-blocking read/write; the *ptq*(7) driver allows a device to be opened with O_NDELAY flag set. The read/write calls on such a device do not block.
- The terminal identification on the IP network can be given to the ptq driver. This identification must be the first four characters of the remote server IP name.
  Example: the cndsa command uses this identification: #<open7_id><terminal_id> where open7_id is the first four characters of the OPEN 7 hostname.

There is an additional internal table, described by the structure *ptqafftab* declared in *ptq.h*:

```
struct ptqafftab {
     int delay_flag;
        /* set when O_NDELAY flag is used for open() ptq */
     struct proc *pesel;
        /* possible proc waiting for exception select */
     int id_timeout;
     int second_timeout;
     char calling_mbx(9) ; /* the 4th argument of VCOPMG */
     char addr_ip(10) ; /* Terminal address */
}
```

Initiation of a connection

Parameters used at connection time are as follows:

Set implicitly by the driver:

- half or full duplex exchanges,
- letter size: 2048 bytes maximum,
- default values for terminal-type, site, mailbox, user, project, and password can be set (from configuration values).

Set explicitly by the user (*ioctl* (2)):

These values are described in the structure *ptctl* declared in *ptq.h*. The values of the *ptctl* fields are retrieved from the kernel using the *ioctl*(2) command PTGET, and modified in the internal structure using the *ioctl*(2) command PTSET.

- terminal type (see legal values in *termio*(7)),
- name of the site on which the session is to be initiated,
- site mailbox name,
- session user,
- project under which the session is to operate; if omitted, the default project in the target site's catalog is taken,
- billing under which the session is to operate; when omitted, the default billing in the target site's catalog is taken,
- the user's password,
- automatic translation mode or not (binary mode), depending on the negotiated values (EBCDIC to ASCII).

The passthrough quick device is a character device (major 27 to 34). The system calls accepted are *open*(2), *close*(2), *read*(2), *write*(2), *ioctl*(2), and *select*(2). On the first open, the internal structures are initialized to the following default values:

- terminal type: TTU8124
- mode: raw mode
- mailbox: "IOF"
- system: spaces (main site)
- project: PTPROJECT_INIT configuration parameter value
- billing: PTBILLING_INIT configuration parameter value
- user: PTUSER_INIT configuration parameter value suffixed by device minor number.
- password: PTPASSWORD_INIT configuration parameter value. Not to be used.

mode is the only field of the *ptctl* structure that can be changed during the whole connection. After connection opening, changes to the field *mode* are accepted but the *ioctl*(2) PTSET returns -1.

There are two ways of opening a VCAM session with *ptq*(7):

- the connection can be opened using a specific *ioctl*(2) command, PTOPENCNX

- or the connection phase with VCAM can be delayed to the first read system call in the device.

NOTE: For a connection to IOF, you should always use read. This is because IOF always has the turn immediately after opening. The driver will sleep indefinitely on write if you open in write, because it is not your turn. The restriction does not apply, however, if you opened the driver with the O_NDELAY flag.

APPLICATION NOTES

The following *ioctl*(2) system calls are available:

To read or to set the Connection values

```
#include <sys/tty.h>
#include <sys/ptq.h>

struct ptctl ptctl;
ioctl(fildes, command, &ptctl);
```

with the commands:

PTGET	Retrieve the default values.
PTSET	Set the default values.

Retrieve Driver internal structure *pttab*

```
#include <sys/tty.h>
#include <sys/ptq.h>

struct pttab pttab;
ioctl(fildes, command, &pttab);
```

with the command:

PTGETTAB	Retrieve the contents of the internal structure.

In normal implementation, the command PTGETTABQ should be used.

Retrieve Driver internal structure *ptqtab*

```
#include <sys/tty.h>
#include <sys/ptq.h>

struct ptqtab ptqtab;
ioctl(fildes, command, &ptqtab);
```

with the command:

PTGETTABQ	Retrieve the contents of the internal structure.

See *ptq.h* file for details.

To read or set mode values

```
#include <sys/tty.h>
#include <sys/ptq.h>

struct pt_mode pt_mode;
ioctl(fildes, command, &pt_mode);
```

with the commands:

PTGETMODE	Retrieve the mode.
PTSETMODE	Set the mode.

To open a VCAM session

```
#include <sys/ptq.h>

ioctl (fildes, command, 0)
```

with the command:

PTOPENCNX	Open connection.

To send a telegram through VCAM

```
#include <sys/ptq.h>

struct ptqtlg pteleg;
ioctl (fildes, command, &pteleg)
```

with the command:

PTSENDTLG	Send telegram.

***ptctl* structure**

```
#define PTIOC ('P'<<8)
#define PTGET (PTIOC|1)
#define PTSET (PTIOC|2)
```

The *ptctl* structure returns to the user or to the driver the common values that enable VCAM to establish a connection with a correspondent.

```
struct ptctl { /* structure for IOCTL */
  unsigned char system[5];   /* node name */
  unsigned char mailbox[9];  /* mailbox name */
  unsigned char billing[13]; /* billing */
  unsigned char user[13];    /* user name */
  unsigned char project[13]; /* project name */
  unsigned char password[13];/* password */
  char binary; /* binary transfer - no translation-patch binary */
  char rfu2;   /* for alignment purposes - patch binary */
  int vcstat;  /* last status in VCAM interface */
  int mode;    /* connection mode */
  unsigned char rc_edit[40]; /* last edited GCOS7 return code */
  unsigned int rc;           /* last GCOS 7 return code */
  int onegsz;                /* size negotiated */
  int f15model;              /* terminal model */
};
```

system	name of the site to which the connection is to be established. An empty string enables connection to the local site. Maximum length 4 bytes, in upper case.
mailbox	name of site mailbox, in upper case. Maximum length 8 bytes.
billing	billing for the session, in upper case. When omitted (empty string), the default billing is applied. Maximum length 12 bytes.
user	session user, in upper case. Maximum length 12 bytes.
project	session project, in upper case. When omitted (empty string), the default project for the user is applied. Maximum length 12 bytes.
password	user password. Maximum length 12 bytes.
binary	transparent mode, without automatic translation (EBCDIC to/from ASCII) with connections in EBCDIC code.
vcstat	last VCAM status known by the driver (PTGET command).

mode — sets the internal mode in the driver. The following *mode* values are used:

```
#define PT_CONS 0x01 /* conserve connection after close */
#define PT_STRU 0x10 /* struct pt_stru in first 8 bytes*/
```

Conserve Connection after last close (Flag PT_CONS).

The flag PT_CONS causes the VCAM connection to be to conserved after the last close in the device driver. At the next open in the device, the link with the previous VCAM connection is re-established.

Structured read (Flag PT_STRU)

If this flag is set, at each *read*(2) system call the first eight bytes pass the structure pt_stru. This structure contains two fields:

```
struct pt_stru {
  int state;    /* internal state */
  int breakrcv; /* break received */
}
```

int state — internal status of the device driver. In the case of a half duplex connection, the analysis of the PTTURN flag is useful to determine when the application has the turn (right to write in VCAM).

int breakrcv — if set, means that a telegram has been received and sent to the application. The data received by *read* (2) contains the telegram data. In IOF for synchronous terminals, telegrams are used to convey the content of the 25th line.

Raw mode only is available. Each read/write system call corresponds to a read/write at the VCAM interface. The message sizes are limited to those offered by VCAM.

The following characters have a special meaning if they are placed as the first character in a *write* (2) system call:

- `0x7f` (ASCII DEL) or `0x1C` (ctrl |)

  Sends a break. Note that a break can cause data loss.

***pt_mode* structure**

```
#define PTGETMODE  (PTIOC|4)
#define PTSETMODE  (PTIOC|5)

   struct pt_mode
     {/* structure for ioctl PTGETMODE and PTSETMODE */
     int state; /* internal state, PTGETMODE only */
     int mode;
     };
```

state defines the internal status of the driver (PTGETMODE command). This is the same field as that declared in the structure *pt_stru*. The most important flags are described below:

PTTURN	UNIX has the turn; half duplex DSA session.
PTCREDIT	credit event received from VCAM; half or full duplex session.
PTTRUNC	truncated data received in the driver buffer.
PTBRKRCV	telegram data in driver buffer.
PTWEND	VCAM connection being terminated; the process must close the device.
PTTOCLOSE	VCAM connection closed; the process must close to device.
PTEBCDIC	the VCAM connection indicates EBCDIC code; if the binary field (*ptctl* structure) is not set, the data will be translated from ASCII to EBCDIC in case of *write*(2) or from EBCDIC to ASCII in case of *read*(2).
PTOPENACK	connection initiation has been acknowledged by VCAM.
PTDMDTURN	an interrupt *demand turn* has been received from VCAM.
PTABNTERM	an interrupt *abnormal term* has been received from VCAM.

mode sets the internal mode. See *field mode* in the ptctl struct.

ptqtab **structure**

This structure defines the internal driver structure used to support the connection with VCAM.

CONFIGURATION

Major device 27 to 34

The first 256 minor entries are attributed to major 27. The following minor entries are attributed to major 28 through major 34 (maximum 2048 entries).

The number of minor entries is given by *define NPTQ* in *sys/config.h*. The default value is 256 (#define NPTQ 256).

FILES

`/dev/dsa/*`

SEE ALSO

setdsa(1), cndsa(1), formc(1), sys/pt.h, sys/ptq.h, config.h

9. pty(7)

NAME

pty - pseudo-terminal driver

SYNOPSIS

```
#include <fcntl.h>
#include <sys/termio.h>
open("/dev/ttypn", mode);
open("/dev/ptypn", mode);
```

DESCRIPTION

The *pty* driver provides the remote application accessed through *telnet* with the same interface as the standard interface *tty*.

It allows this application to view the remote terminal used for the connection as a standard *tty*. Thus it performs virtual terminal functions.

The *pty* driver provides support for a pair of devices jointly known as a pseudo-terminal. The two devices comprising a pseudo-terminal are known as a controller and a slave. The slave device provides an interface identical to that described in *termio*(7). Instead of having a hardware interface and associated hardware which supports the terminal functions, the functions are implemented by another process manipulating the controller device of the pseudo-terminal.

The controller and the slave devices of the pseudo-terminal are closely connected. Any data written to the controller device is passed to the slave device as input, as though it had been received from a hardware interface. Any data written on the slave terminal can be read from the controller device.

The standard set of *termio ioctl*'s are supported by the slave device.

None of the bits in the *c_cflag* word have any effect on the pseudo-terminal.

There is no notion of parity on a pseudo-terminal, so none of the flags in the *c_iflag* word that control the processing of parity errors have any effect. Similarly, there is no notion of a break, so none of the flags that control the processing of breaks, and none of the *ioctl*'s that generate breaks has any effect.

Input flow control is automatic: a process that attempts to write to the controller device will be blocked if too much unconsumed data is buffered on the slave device. The input flow control provided by the IXOFF flag in the *c_iflag* word is not supported.

The delays specified in the *c_oflag* word are similarly not supported.

ioctl Calls (Master Device Only)

The following *ioctl* calls apply only to the master device of the pseudo-terminal:

TIOCPKT
The argument is a pointer to an *int*. If the value of the *int* is non-zero, packet mode is enabled; If the value of the *int* is zero, packet mode is disabled.

When a pseudo-terminal is in packet mode, each subsequent *read* (2) from the controller device returns data written on the slave device preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more bits:

TIOCPKT_FLUSHREAD
whenever the read queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE
whenever the write queue for the terminal is flushed.

TIOCPKT_STOP
whenever output to the terminal is stopped using ^S.

TIOCPKT_START
whenever output to the terminal is restarted.

TIOCPKT_DOSTOP
whenever XON/XOFF flow control is enabled after being disabled; it is considered enabled when the IXON flag in the c_iflag word is set.

TIOCPKT_NOSTOP
whenever XON/XOFF flow control is disabled after being enabled. While this mode is in use, the presence of control status information to be read from the master side may be detected by a select for exceptional conditions.

This mode is used by *rlogin*(1) and *rlogind*(1M) for a remote login with remote echo, and local ^S/^Q flow control. It is accompanied by correct back-flushing of output when interrupts occur. It can be used by other similar programs.

TIOCREMOTE
The argument is a pointer to an *int*. If the value of the *int* is non-zero, remote mode is enabled; If the value of the *int* is zero, remote mode is disabled.

This mode can be enabled or disabled independently of packet mode.

When a pseudo-terminal is in remote mode, input to the slave device of the pseudo-terminal is flow controlled and input is not edited (regardless of the mode of the slave side of the pseudo-terminal). Each write to the controller device produces a record boundary for the process reading the slave device.

In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an EOF character. Note that this means that a process writing to a pseudo-terminal controller in remote mode must keep track of line boundaries, and write only one line at a time to the controller.

If, for example, it were to buffer up several NEWLINE characters and write them to the controller with one *write* (), it would appear to a process reading from the slave as if a single line containing several NEWLINE characters had been typed.

Remote mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

FILES

`/dev/pty[p-s][0-9a-f]`	pseudo-terminal controller devices
`/dev/tty[p-s][0-9a-f]`	pseudo-terminal slave devices
`/usr/include/sys/pty.h`	symbolic definitions
`/makespix/cf/config.h`	number of pseudo-terminal pairs supported (NPTY)

APPLICATION NOTES

The default number of pseudo-terminal pairs supported is 16 (#define NPTY 16), and the corresponding character special files are */dev/ptyp[0-9a-f]* and */dev/ttyp[0-9a-f]*.

The major numbers of slave and controller devices are 14 and 15 respectively.

RESTRICTIONS

It is apparently not possible to send an EOT by writing zero bytes in TIOCREMOTE mode.

SEE ALSO

rlogin(1), termio(7), rlogind(1M)

9. pwck(1M), grpck(1M)

NAME

pwck, grpck - password/group file checkers

SYNOPSIS

```
pwck [file]
grpck [file]
```

DESCRIPTION

pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is */etc/passwd*.

grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is */etc/group*.

FILES

/etc/passwd

/etc/group

NOTES

Names longer than 14 characters are not supported.

C2 Secure Environment
In a Secure environment, this command can be used only by the superuser with *authentication* subsystem privilege.

SEE ALSO

group(4), passwd(4)

DIAGNOSTICS

Group entries in */etc/group* with no login names are flagged.

9. pwd(1)

NAME

pwd - print working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

Index

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 OPEN 7 Command Reference Manual (H to P) Operating System: Subsystems

Reference N° :	47 A2 83US 01	**Date:**	**April 1997**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : _____ Date : _____

COMPANY : _____

ADDRESS : _____

_____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
[_ _] : The latest revision will be provided if no revision number is given.		

NAME: ____ Date: ____________

COMPANY: ____

ADDRESS: ____

____

PHONE: ______________________________ FAX: ____________________

E-MAIL: ____

For Bull Subsidiaries:

Identification: ____

For Bull Affiliated Customers:

Customer Code: ____

For Bull Internal Customers:

Budgetary Section: ____

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 83US 01