

# Appendix A

# Man Pages

man whatis

whatis(1)

# Typical Man Page Sections

Section	Contents
1	User commands
2	System calls
3	C library functions
4	Devices and network interfaces
5	File formats
6	Games and demos
7	Environments, tables, and troff
8	macros System maintenance

# Man Page Layout

**HEADER:** a title for the individual man page

**NAME:** a one-line summary

**SYNOPSIS:** describes usage

**AVAILABILITY:** indicates availability on the system

**DESCRIPTION:** discusses what the command or function does

**RETURN VALUES:** the return values if applicable

**ERRORS:** summarizes errno values and conditions for errors

**FILES:** lists the system files that the command or function uses

**SEE ALSO:** lists related cmds/additional sects of the manual

**ENVIRONMENT:** lists relevant environment variables

**NOTES:** provides info on unusual usage/implementation features

**BUGS:** lists known bugs and caveats



# More

--More-- (41%)

Indicates percentage of man page yet to be displayed.

To get next screenful, hit space bar

To get next line, hit return

To quit hit q

# man intro Example

man intro

displays man page for first  
available section of a  
command (in this case the  
section 1 intro man page  
intro(1))

# -k Option

man -k intro

displays information about all sections of the man pages:

Intro Intro (1) - introduction to commands and application programs

Intro Intro (1m) - introduction to maintenance commands and application programs

Intro Intro (2) - introduction to system calls and error numbers

Intro Intro (3) - introduction to functions and libraries

# -s Option

`man -s 2 intro`

displays only section 2 man  
page of intro

# -a Option

`man -a intro`

Displays man pages for all intro sections

# Synopsis

The synopsis entry for system calls indicates the function prototype and library functions to include when using that command. For example, for `write(2)`:

## SYNOPSIS

```
#include <unistd.h>
```

```
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

# File Descriptor I/O

```
#include <unistd.h>
#include <string.h>
#define MYMSG "hello world\n"
int bytes;
bytes=write(STDOUT_FILENO, MYMSG,
           strlen(MYMSG));
```

Note that if `unistd.h` were left out of the above example, `STDOUT_FILENO` would be undefined.

# File Descriptors

Standard file descriptors:

STDIN_FILENO	standard input	0
STDOUT_FILENO	standard output	1
STDERR_FILENO	standard error	2

# File Pointer I/O

```
#include <stdio.h>
```

```
fprintf(stdout, "hello world"\n);
```

# Uninitialized File Pointer

```
#include <unistd.h>  
char *mybuf;  
size_t nbytes;  
write(STDOUT_FILENO, mybuf, nbytes);
```

Code will likely produce uninitialized pointer error.

# Garbage

```
#include <unistd.h>
char mybuf[100];
size_t nbytes;
write(STDOUT_FILENO, mybuf, nbytes);
```

Code would probably print out garbage.  
Note that write function returns value of type `ssize_t`. Always capture it and test for errors (i.e., returns `-1` with `errno` set to error type).

# Command Level write

```
write(1)
```

```
write annie
```

The above command opens a write session to account annie. Everything typed subsequent appears on annie's screen. Terminate with ctrl-d. Of course, annie must give permission.

# who

who -H

Prints out those who are logged on and on which terminal line.

# Command Definitions

Print out command definitions with:

```
stty -a
```

# Command-Line Arguments

Consist of:

- Options
- Option Arguments
- Operands

# Options/Option Arguments

- Option name consists of single alphanumeric number
- All options are preceded by “-”
- Options with no option-arguments may be grouped after a single “-”
- First option-argument following an option is preceded by a tab or space character
- Option-arguments cannot be optional
- Groups of option-arguments following an option must either be separated by commas or by tab or space characters and quoted (-o xxx,z,yy or -o “xxx z yy”)
- All options must precede operands on the command line
- “--” may be used to indicate the end of options
- Relative order of options is important
- When preceded or followed by a space character, “-” should only be used to indicate standard input

# User Command Options

ls (1)

## SYNOPSIS

ls [-abcCdfFgiLmnopqrRstux1] [names]

Each letter is described either in the DESCRIPTION or OPTIONS section of the man page.

# apropos/whatis/which

apropos cmd

lists all man page entries where phrase “cmd” appears

whatis cmd

gives a one line synopsis of the cmd

which cmd

gives a complete pathname of the file named “cmd”  
be careful that it is in the section you want

# find

## SYNOPSIS

```
find path ... [operand-expression ...]
```

The following command lists all files in the current directory that end in .c:

```
find -name "*.c" -print
```

# Compilation

```
gcc -o myfile myfile.c
```

Compiles myfile.c and gives executable name myfile

Without -o option, a.out is produced by default

# Appendix sections A.2-A.6

Appendix sections A.2-A.6 cover compilation, makefiles, linking and libraries, and debugging aids. You should already be familiar with these concepts. If not, read the sections on your own.

# UNIX Shells

- C shell (csh),
- Bourne Shell (sh)
- KornShell (ksh)

# Startup Commands

- C shell takes startup commands from `.cshrc`
- Bourn shell and KornShell take startup commands from `.profile`

# Common Environment Variables

- HOME=home directory
- SHELL=shell of user
- PATH=pathnames of directories to be searched when locating commands to be executed
- LOGNAME=login name
- USER=user name
- MANPATH=pathnames of directories to be searched when locating man pages
- DISPLAY=display to use for the X Window System

# Environment Variable Usage

Environment variables are normally uppercase. Frequently used environment variables are initialized by the user in the startup file.

# setenv/export

Try setenv (C shell) and export (KornShell) to determine the type of your shell.

```
setenv EDITOR vi  
export EDITOR=vi
```

# env Command

## SYNOPSIS

```
env [-i] [name=value] ... [utility [argument...]]
```

Modifies current environment with each name set to specified value and then executes utility with modified environment.

env alone sends environment list to standard output

# Printing Environment List

```
#include <stdlib.h>
#include <stdio.h>
extern char **environ;
void main(int argc, char *argv[])
{
    int i;
    for(i=0;*(environ+i)!=NULL;i++)
        printf("%s\n",*environ+i);
    exit(0); }
```

Or alternatively, just type env.