ulimit() — System Call (libc)

Get/set limits for a process
#include <ulimit.h>
long ulimit (*command* **[,** *blocks*^**])**
int *command***,** *blocks*^**;**

The system call **ulimit()** retrieves or sets limits on what a process can do. *command* indicates what you want it to do, as follows:

UL_GETFSIZE
Return the maximum size, in blocks, of a file that the current process can create.

UL_SETFSIZE
Limit to *blocks* the size of any regular file that any process can create. A process may decrease this limit, but only a process owned by the superuser **root** can increase it.

UL_GMEMLIM
Return the current process's break value. For details on the break value, see the Lexicon entry for **brk()**.

UL_GDESLIM
Return the maximum number of files that this process can open.

Each of the above commands is defined in the header file **ulimit.h**. When called to execute the command **UL_SETFSIZE**, **ulimit()** requires a second integer argument; when called to execute any other command, **ulimit()** takes only one argument.

If all goes well, **ulimit()** returns a non-negative value. **ulimit()** fails if any of the following occur:

- A process owned by someone other than the superuser **root** attempted to increase its file-size limit. **ulimit()** returns -1 and sets **errno** to **EPERM**.
- The first argument to **ulimit()** was something other than one of the above-named values. **ulimit()** returns -1 and sets **errno** to **EINVAL**.

See Also

brk(), libc, ulimit.h

Notes

ulimit() does not fail *per se* if you invoke it with option **UL_SETFSIZE** and do not supply a second argument. However, doing so will (or should) crash the process. *Caveat utilitor.*

ulimit.h — Header File

Define manifest constants used by system call ulimit()
#include <ulimit.h>

The header file **ulimit.h** defines manifest constants used with the system call **ulimit()**.

See Also

header files, ulimit()

umask — Command

Set the file-creation mask
umask [*OOO*]

The *file-creation mask* modifies the default mode assigned to each file upon creation. The mode sets the permissions granted by the file's owner, plus other important information about a file.

The command **umask** sets the default file-creation mask to *OOO*, which are three octal numerals. If invoked without an argument, **umask** prints the current file-creation mask in octal.

Note that zero bits in the mask correspond to permitted permission bits in the target, and that execute permission cannot be enabled via any setting of *mask*. See the Lexicon entries for **umask()** and **chmod** for further details on file mode. The shell executes **umask** directly.

Example

Setting *mask* to octal 022 (i.e., 000 010 010) causes a file created with mode octal 0666 to actually have permissions of

```
rw- r-- r--
```

Setting *mask* to zero (i.e., 000 000 000) causes a file created with mode octal 0666 to actually have permissions of

```
rw- rw- rw-
```

See Also

chmod, commands, ksh, sh, umask()

umask() — System Call (libc)

Set file-creation mask
#include <sys/stat.h>
int umask(*mask***)**
int *mask***;**

umask() allows a process to restrict the mode of files it creates. Commands that create files should specify the maximum reasonable mode. A parent (e.g. the shell **sh**) usually calls **umask()** to restrict access to files created by subsequent commands.

mask should be constructed from any of the permission bits found by **chmod()** (the low-order nine bits). When a file is created with **creat()** or **mknod()**, every bit set in the *mask* is zeroed in *mode*; thus, bits set in *mask* specify permissions that will be denied.

umask() returns the old value of the file-creation mask.

Example

Setting *mask* to octal 022 (i.e., 000 010 010) causes a file created with mode octal 0666 to actually have permissions of

```
rw- r-- r--
```

Setting *mask* to zero (i.e., 000 000 000) causes a file created with mode octal 0666 to actually have permissions of

```
rw- rw- rw-
```

See Also

creat(), libc, mknod(), sh, stat.h
POSIX Standard, §5.3.3

Notes

A file's default permission cannot be set to execute regardless of the value of *mask*.

umount — Command

Unmount file system
/etc/umount *special*

umount unmounts a file system *special* that was previously mounted with the **mount** command.

The script **/bin/umount** calls **/etc/umount**, and provides convenient abbreviations for commonly used devices. For example, typing

```
umount f0
```

executes the command

```
/etc/umount /dev/fha0
```

The system administrator should edit this script to reflect the devices used on your specific system.

Files

/etc/mtab — Mount table
/dev/*
/bin/umount — Script that calls **/etc/umount**

See Also

clri, commands, fsck, icheck, mount

Diagnostics

Errors can occur if *special* does not exist or is not a mounted file system.

umount() — System Call (libc)

Unmount a file system
#include <sys/mount.h>
umount(*filesystem***)**
char ****filesystem*;**

umount() is the COHERENT system call that unmounts a file system. *filesystem* names the block-special file through which the file system is accessed. Note that this must have been previously mounted by a call to **mount()**, or the call will fail.

See Also

libc, mount()

unalias — Command

Remove an alias
unalias *alias ...*

The command **unalias** is built into the Korn shell **ksh**. It removes each *alias*.

See Also

alias, commands, ksh

uname — Command

Print information about COHERENT
uname [-amnrsv]
uname [-S *systemname* **]**

The command **uname** prints information about the current implementation of COHERENT. It recognizes the following options:

-a Print all information.

-m Print the machine on which this implementation of COHERENT is running. This always defaults to the Intel 80386.

-n Print the name of your system, as set in the file **/etc/uucpname**.

-r Print the release of your copy of COHERENT.

-s Print the system name.

-S Change the system name. *systemname* is restricted to eight characters.

-v Print the version of COHERENT.

Example

The following script uses **uname** to implement a version of the Sun OS command **hostname**. It is by Cy Schubert (cschuber@bcsc02.gov.bc.ca):

```
#!/bin/sh -
# hostname - display or change the name of the host system
case $# in
    0)    uname -n;;
    1)    uname -S $1;;
    *)    echo Usage: hostname [new_hostname]
          exit 1;;
esac
```

See Also

commands

uname() — System Call (libc)

Get the name and version of COHERENT
#include <sys/utsname.h>
int uname(*name***)**
struct utsname ***name*;**

The COHERENT system call **uname()** identifies the current release of the COHERENT operating system. It writes its output into the structure pointed to by *name*. This must be of type **utsname**, which has the following members:

```
char sysname[SYS_NMLN];            /* system name */
char nodename[SYS_NMLN];           /* UUCP node name */
char release[SYS_NMLN];            /* current release */
char version[SYS_NMLN];            /* current version */
char machine[SYS_NMLN];            /* hardware */
```

uname() returns a non-negative value upon success. If something went wrong, i.e., *name* points to an invalid address, **uname()** returns -1 and sets **errno** to an appropriate value.

See Also

libc, utsname.h
POSIX Standard, §4.4.1

Notes

The COHERENT implementation of **uname()** conforms to POSIX Standard, which states that **uname()** returns a "non-negative" value upon success. To write portable code, your code must check for a return value that is greater than or equal to zero. It is an error to check for return value equal to zero, because the test works on some systems that adhere to the Standard but not on others.

uncompress — Command

Uncompress a compressed file
uncompress [*file* ... **]**

uncompress uncompressses one or more *file*s that had been compressed by the command **compress**.

Each *file*'s name must have the suffix **.Z**, which was appended onto it by **compress**; otherwise, **uncompress** prints an error message and exits. When **uncompress** has uncompressed a *file*, it removes the **.Z** suffix from that file's name.

If no *file* is specified on the command line, **uncompress** uncompresses matter read from the standard input, and writes its output to the standard output.

Older versions of **uncompress** could only uncompress files that had been compressed with option **-b12** or lower, with **-b12** being the default. The edition of **uncompress** released with COHERENT version 3.1 (and subsequent versions) can handle values up to 16.

See Also

commands, compress, compression, ram, zcat

unctrl.h — Header File

Define macro unctrl()
#include <unctrl.h>

The header file **unctrl.h** defines the macro **unctrl()** which changes a character from a control character to a printable character.

See Also

header files

ungetc() — STDIO Function (libc)

Return character to input stream
#include <stdio.h>
int ungetc (*c*, *fp***)**
int *c***; FILE ***fp***;**

ungetc() returns the character *c* to the stream *fp*. *c* can then be read by a subsequent call to **getc()**, **gets()**, **getw()**, **scanf()**, or **fread()**. No more than one character can be pushed back into any stream at once. A call to **fseek()** will nullify the effects of a previous **ungetc()**.

ungetc() normally returns *c*. It returns **EOF** if the character cannot be pushed back.

Example

For an example of this function, see **fgetc()**.

See Also

fgetc(), getc(), libc
ANSI Standard, §7.9.7.11
POSIX Standard, §8.1

union — C Keyword

Multiply declare a variable

A **union** defines an area of storage that can accept any one of several types of data. In effect, it is a multiple declaration of a variable. For example, a **union** may be declared to consist of an **int**, a **double**, and a **char ***. Any one of these three elements can be held by the **union** at a time, and will be handled appropriately by it. For example, the declaration

```
union {
      int number;
      double bignumber;
      char *stringptr;
} example;
```

allows **example** to hold either an **int**, a **double**, or a pointer to a **char**, whichever is needed at the time. All of these have the same address. The elements of a **union** are accessed like those of a **struct**: for example, to access **number** from the above example, type **example.number**.

unions are helpful in dealing with heterogeneous data, especially within structures; however, you are responsible for keeping track of what data type the **union** is holding at any given time. Passing a **double** to a **union** and then reading the **union** as though it held an **int** will yield results that are unpredictable, and probably unwelcome.

Example

For an example of how to use a **union** in a program, see the entry for **byte ordering**.

See Also

C keywords, initialization, struct, structure
ANSI Standard, §3.1.2.5, §3.5.2.1

uniq — Command

Remove/count repeated lines in a sorted file
uniq [-cdu] [-*n*] [+*n*] [*infile*[*outfile*]]

uniq reads input line by line from *infile* and writes all non-duplicated lines to *outfile*. The input file must be sorted. **uniq** uses the standard input or output if either *infile* or *outfile* is omitted. The following describes the available options:

-c Print each line once, discarding duplicate lines; before each line, print the number of times it appears within the file.

-d Print only lines that are duplicated within the file; print each line only once; do not print any counts.

-u Print only lines that are *not* duplicated within the file.

uniq by default behaves as if both **-u** and **-d** were specified, so it prints each unique line once.

Optional specifiers allow **uniq** to skip leading portions of the input lines when comparing for uniqueness.

-*n* Skip *n* fields of each input line, where a field is any number of non-white space characters surrounded by any number of white space characters (blank or tab).

+*n* Skip *n* characters in each input line, after skipping fields as above.

See Also

comm, commands, sort

unistd.h — Header File

Define constants for file-handling routines
#include <unistd.h>

The header file **unistd.h** defines standard routines used by the UNIX and UNIX-like operating systems. It prototypes many commonly used functions, and declares manifest constants used when checking file access, setting the seek pointer, and locking files.

See Also

access(), fseek(), header files, lockf()
POSIX Standard, §2.9

units — Command

Convert measurements
units [-u]

units is an interactive program that tells you how to convert one unit of measurement into another. It prompts you for two quantities with the same dimension (e.g., two measurements of weight, or two of size). It first prints the prompt “You have:” to ask for the unit you wish to convert from, and then prints the prompt “You want:” for the unit you wish to convert to.

Example

The following example returns the formula for convert fortnights into days:

```
You have: fortnight
You want: days
* 14
/ 0.071428
```

The following fundamental units are recognized: **meter**, **gram**, **second**, **coulomb**, **radian**, **bit**, **unitedstatesdollar**, **sheet**, **candle**, **kelvin**, and **copperpiece** (shillings and pence).

A quantity consists of an optional number (default, 1) and a dimension (default, none). Numbers are floating point with optional sign, decimal part and exponent. Dimensions may be specified by fundamental or derived units, with optional orders. A quantity is evaluated left to right: a factor preceded by a ‘/’ is a divisor, otherwise it is a multiplier. For example, the earth’s gravitational acceleration may be entered as any of the following:

```
      9.8e+0 m+1 sec-2
      32 ft/sec/sec
      32 ft/sec+2
```

British equivalents of US units are prefixed with **br,** e.g., **brpint**. Other units include **c** (speed of light), **G** (gravitational constant), **R** (gas-law constant), **phi** (golden ratio) % (1/100), **k** (1,024), and **buck** (United States dollar).

/usr/lib/units is the ASCII file that contains conversion tables. The binary file **/usr/lib/binunits** may be recreated by using the **-u** option.

Files

/usr/lib/units — Known units
/usr/lib/binunits — Binary encoding of units file

See Also

bc, commands, conv

Diagnostics

If the ASCII file **/usr/lib/units** has changed more recently than the binary file **/usr/lib/binunits**, **units** prints a message and regenerates the binary file before it continues; this can take up to a few minutes, depending upon the speed of your system.

The error message “conformability” means that the quantities are not dimensionally compatible, e.g., **m/sec** and **psi**. **units** prints each quantity and its dimensions in fundamental units.

Notes

There are the inevitable name collisions: **g** for gram versus **gee** for Earth’s gravitational acceleration, **exp** for the base of natural logarithms versus **e** for the charge of an electron, **ms** for (plural) meters versus **millisecond,** and, of course, **batman** for the Persian measure of weight rather than the Turkish.

unlink() — System Call (libc)

Remove a file
#include <unistd.h>
int unlink(*name***) char ***name***;**

unlink() removes the directory entry for the given file *name*, which in effect erases *name* from the disk. *name* cannot be opened once it has been **unlink()**’d. If *name* is the last link, **unlink()** frees the i-node and data blocks. Deallocation is delayed if the file is open. Other links to the file remain intact.

Example

This example removes the files named on the command line.

```
#include <unistd.h>
main(argc, argv)
int argc; char *argv[];
{
      int i;

      for (i = 1; i < argc; i++) {
            if (unlink(argv[i]) == -1) {
                  printf("Cannot unlink \"%s\"\n", argv[i]);
                  exit(EXIT_FAILURE);
            }
      }
      exit(EXIT_SUCCESS);
}
```

See Also

libc, link(), ln, remove(), rm, rmdir, unistd.h
POSIX Standard, §5.5.1

Diagnostics

unlink() returns zero when successful. It returns -1 if *file* does not exist, if the user does not have write and search permission in the directory containing *file*, or if *file* is a directory and the invoker is not the superuser.

unpack — Command

GNU utility to uncompress files
unpack [-cfhLrtvV] [*file ...* **]**

unpack uncompresses each *file* named on its command line. Each *file* must have been compressed by the COHERENT commands **gzip** or **compress**, or by the UNIX command **pack**. If no *file* appear on its command line **unpack** uncompresses what it reads from the standard input.

unpack is a link to the command **gunzip**. For details on its command-line options, see the Lexicon entry for **gunzip**.

See Also

commands, gzip, gunzip

unset — Command

Unset an environment variable or shell function
unset *environmental_variable*
unset -f *shell_function*

The command **unset** unsets an environmental variable or shell function.

When used with the option **-f**, **unset** unsets the shell function named on the command line. This option applies only to the Bourne shell **sh**.

When used without the option **-f**, **unset** unsets the environmental variable named on the command line. This version of the command applies to both the Bourne shell **sh** and the Korn shell **ksh**.

See Also

commands, environmental variables, ksh, sh

unsigned — C Keyword

Data type

unsigned tells the compiler to treat the variable as an unsigned value. In effect, this doubles the largest absolute value that that type can hold, and changes the lowest storage value to zero.

See Also

C keywords, data type
ANSI Standard, §6.2.1.2

until — Command

Execute commands repeatedly
until *sequence1* **[do** *sequence2* **] done**

The shell's **until** loop executes the commands in *sequence1*. If the exit status is nonzero, the shell then executes the commands in the optional *sequence2* and repeats the process until the exit status of *sequence1* is zero. Because the shell recognizes a reserved word only as the unquoted first word of a command, both **do** and **done** must occur either unquoted at the start of a line or preceded by ';'.

The shell commands **break** and **continue** may be used to alter control flow within an **until** loop. The contruct **while** has the same form as **until** but the sense of the test is reversed.

The shell executes **until** directly.

See Also

break, commands, continue, ksh, sh, test, while

unzip — Command

Un-zip a zipped archive
unzip *archive* **[-cfpux** *file ...***] [-ltvz] [-anojqUV]**

The command **unzip** extracts files from a zipped archive. It recognizes the following command-line options:

-c [*file ...***]**
Extract *file*, but write them to the standard output instead of to disk.

-f [*file ...***]**
"Freshen" files: Extract *file* from *archive* and write it to disk, but do so only if the file in the archive is newer than the file on disk. Do not create new files.

-l List the contents of the archive, short format.

-p [*file ...* **|** *command***]**
Extract each *file* and pipe it to *command*.

-t Test the integrity of *archive*.

-u [*file ...***]**
Update each *file* within the archive. Create the file if necessary.

-v List files, verbose format.

-x [*file ...***]**
Extract each *file* from default. If no *file* argument is given, extract all files. This is the default.

-z Display archive's comments, if any.

The following modify the behavior of the options:

-a Convert text from MS-DOS format to UUCP format.

-j Ignore ("junk") paths; do not make directories.

-n Never overwrite existing files.

-o Overwrite files without prompting.

-q Quiet mode.

-qq Quieter mode.

-U Do not convert file names to lower-case letters.

The following example extracts file **ReadMe** from archive **data1**:

```
unzip data1 ReadMe
```

The next example extracts all files from archive **foo.zip** and pipes them to the pager **more**:

```
unzip -p foo | more
```

The final example "freshens" files on disk from the contents of **foo.zip**. Files are overwritten without prompting:

```
unzip -fo foo
```

Notes

commands, compress, gunzip, gzip, uncompress, zip

Notes

Do not confuse this command with **gunzip**. Archives made by **gzip** may not be extractable by **unzip**.

upac — Command

De-fragment a file system without sorting
upac *raw_device*

Command **upac** uses de-fragments file system *raw_device* without sorting it by access date. Rather, it orders files by i-node number.

See Also

commands, dpac, fmap, fsck, qpac, spac

Notes

upac is a link to the command **dpac**.

upac was written by Randy Wright (rw@rwsys.wimsey.bc.ca).

update — System Administration

Update file systems periodically
/etc/update

update periodically calls **sync** to write to the disk all file system data that are in memory. It never exits.

The initialization command file **/etc/rc** normally executes **update**. It should not be executed directly.

See Also

Administering COHERENT, init, sync

uproc.h — Header File

Definitions used with user processes
#include <sys/uproc.h>

uproc.h defines the constants and structures used by routines that manage user processes.

See Also

header files

USER — Environmental Variable

Name user's identifier
USER=*user_identifier*

The environmental variable **USER** names your login identifier. For example, if your login identifier is **fwb**, then by typing **set** you will see the entry **USER=fwb**. **USER** is set by **login**.

See Also

environmental variables, ksh, login, LOGNAME, sh

Using COHERENT — Overview

For an ordinary user — that is, one who neither administers the COHERENT system nor writes programs for it — using COHERENT mainly involves issuing commands to the COHERENT system.

The Lexicon entry **commands** names every command that comes with the COHERENT system. The commands are grouped by function. You should look carefully at the shell commands — that is, the commands that work closely with the shell to help you control the execution of other commands. What other groups you study will depend on just what you want to do with your COHERENT system.

Pay particular attention to the Lexicon entries for the commands **sh**, **ksh**, and **vsh**. These introduce the *shells* — that is, the programs with which you can issue commands to COHERENT. Each has its own syntax; **ksh** and **sh** in fact implement fully flown programming languages on their own.

vsh is a visual shell, and is especially useful to beginners. It uses a visual interface and drop-down menus to make it easy for you to issue commands without having to remember convoluted command syntax. The Lexicon entry for **vsh** describes it, and how you can customize it for yourself.

The Lexicon entry for **MS-DOS** compares COHERENT with MS-DOS, and describes how they differ. It also gives a table of COHERENT equivalents to commonly used MS-DOS commands. If you are used to using MS-DOS, you should find this useful.

The follow commands help you to find information about your system:

apropos
  This command searches the description of each Lexicon article for a keyword that you enter. In this way, you can quickly find which articles discuss a given topic, such as "printer" or "modem".

help This command displays a brief summary of each Lexicon article, by name.

man This command displays Lexicon articles on your screen, by name.

The following three articles introduce files that are stored in your home directory. By modifying these files, you can customize your COHERENT account to suit your tastes:

.kshrc Script **$HOME/.kshrc** configures the Korn shell to suit your tastes. You will need to edit this file if you decide to use the Korn shell.

.lastlogin
File **$HOME/.lastlogin** records the date and time you last logged in to your COHERENT system.

.profile Script **$HOME/.profile** holds commands that are executed when a given user logs in to your COHERENT system.

The following Lexicon entries hold technical information that you probably will find useful:

block This defines the size of a "block" on a mass-storage device.

compression
This introduces the subject of compression, and the programs with which you can compress and de-compress files. It also gives a table that describes how to de-compress files based on their default suffices.

environmental variables
This article lists the commonly used environmental variables that are described in the Lexicon. These variables control many of the behaviors of the COHERENT system.

Lexicon
This describes the format of the printed COHERENT manual. It also summarizes changes made to on-line Lexicon pages (the ones that you view via the command **man**) since the manual was last printed.

man This summarizes the **man** macros that are used by the text-formatter **nroff**.

ms This summarizes the **ms** macros that are used by the text-formatter **nroff**.

Finally, the following Lexicon entries define technical terms that are used in this manual:

caveat utilitor
daemon
directory
file
filter
i-node
named pipe
pipe
process
root
sticky bit
superuser
wildcards

For pointers on where to look for information on how to install and modify peripheral devices on your system, such as the keyboard, the hard disk, or a CD-ROM drive, see the Lexicon entry **Administering COHERENT**.

See Also

Administering COHERENT, COHERENT, Programming COHERENT

usleep() — Sockets Function (libsocket)

Sleep briefly
long usleep (*t***)**
long *t***;**

The function **usleep()** sleeps for *t* milliseconds or until it receives a signal.

See Also

libsocket, nap()

Notes

usleep() is included for compatibility with Berkeley socket code. It is the equivalent of the System V Release 4 system call **nap()**.

usrtime — System Administration

Times each user is permitted to log in
/etc/usrtime

File **/etc/usrtime** holds the time, day of the week, and terminal line upon which a given user can log into your COHERENT system. Command **login** reads it to see if a user who is attempting to log in is doing at a permitted time and via a permitted line. If a user is not named in this file, **login** assumes that she can log in at any time, via any line.

usrtime consists of an indefinite number of lines, each with the following format:

*users***:***enable***:***tty***:***weekday***:***time***:***comment*

The following describes each field in detail.

user The login identifiers of the user or users to be restricted. Multiple identifiers must be separated by commas. Each identifier must be defined in **/etc/passwd**. If this field is empty, then the line is a default for every user not specifically named elsewhere in **usrtime**.

The keywords **ALL**, **UUCP**, **SLIP**, and **INTERACTIVE** can also be used in this field, to name categories of users. They are described in detail below.

enable Enable or disable the login (or logins). **NOLOGIN** disables the login; **LOGIN** or an empty field enables it. A range of dates of the form

```
yyyymmdd-yyyymmdd
```

enables logins only during those dates. This field can contain more than one range of dates; if it does, the ranges must be separated by a comma. Prefixing a range of dates with a '!' disables logins between those dates.

tty This field lists the devices via which the user (or users) may log in — usually a **tty** or **com** device. If this field names more than one device, they must be separated by commas. A device name can contain the wildcard character '?'; for details on how this is interpreted, see the Lexicon entry for **wildcards**. If a device is prefixed with a '!', the user cannot log in on that device. If this field is empty, then the user can log in on all devices.

weekday
This field lists the days of the week upon which the the user (or users) can log in. If more than one day is named, they must be separated by commas. Each day is identified by the first three letters of its name. If a weekday is prefixed with a '!', then the users cannot log in on that day. If this field is empty, the users can log in on any day of the week.

time This field gives range of time during which the user (or users) may log in. Time is given in the form:

```
hhmm-hhmm
```

If more than one range is named, they must be separated by commas. Prefixing a range with a '!' forbids the user to log in during between those times. If this field is empty, then the user can log in during any time of the day.

comment
This field holds some commentary, presumably helpful to others who must read this file. **login** ignores this field.

Scope of Entries

A user may be affected by more than entry in this file. The order in which the entries appear is significant.

At the top of the file should appear the entries that are being excluded from restriction. These should include such users as **bin** and **daemon**, plus any ordinary user you wish to exclude from being restricted. The entries for such a users should consist of her (its) name, followed by five colons. Any user named in such an entry is immune to any restrictions that may appear below in this file.

Next should come the global restrictions, that is, restrictions for entire categories of users. As mentioned above, you can use the keywords **ALL**, **UUCP**, **SLIP**, or **INTERACTIVE** to describe users. These keywords have the following meaning:

ALL	All users.
UUCP	All "users" who are UUCP accounts — i.e., whose shell as set in **/etc/passwd** is **/usr/lib/uucp/uucico**.
SLIP	All "users" who are SLIP accounts — i.e., whose shell is **sllogin**.
INTERACTIVE	Users who have an interactive the interactive shell **ksh** or **sh** set at login.

Last should come entries for individual users or clusters of users. These restrictions can be set in addition to those set for categories of users. An entry for an individual users that appears below the global entries will not loosen the restrictions set globally for that user; but it can tighten them.

Note that **login** ignores any restrictions set for the superuser **root**. Finally **login** ignores every line that begins with a '#'. You can use such lines to hold comments.

Example

The following gives an example **usrtime** file:

```
# <user>:<enable>:<tty>:<weekday>:<time>:<comment>
sys,bin,daemon:::::
INTERACTIVE::/dev/com??,/dev/color?:Mon,Tue,Wed,Thu,Fri:0630-1830:
UUCP::/dev/com2l:::UUCP accounts
::::0800-1700:default for anybody not mentioned below
fred,anne:LOGIN:/dev/color?::0830-1630:administration
ivan,marian:LOGIN:/dev/com??:::secretarial staff
catherine:19930401-19931130::::consultant programmer
```

See Also

Administering COHERENT, login

Notes

No line in **usrtime** can exceed 500 characters.

ustat() — System Call (libc)

Get statistics on a file system
#include <sys/types.h>
#include <ustat.h>
int ustat (*device***,** *buffer***)**
dev_t *device***;**
struct ustat **buffer***;**

The COHERENT system call **ustat()** returns information about a mounted file system. *device* names the device upon which the file system is mounted. *buffer* points to a structure of type **ustat**, which contains the following fields:

```
daddr_t        f_tfree;              /* number of free blocks */
ino_t          f_tinode;             /* number of free i-nodes */
char           f_fname[6];           /* name of the file system */
char           f_fpack[6];           /* pack name of the file system */
```

Useful information may not be available for fields **f_fname** and **f_fpack**; in that case, they are initialized to nuls.

ustat() returns zero if all goes well; otherwise, it returns -1 and sets **errno** to an appropriate value. **ustat()** can fail for any of the following reasons:

- *device* does not contain a mounted file system.
- *buffer* points to an illegal address.
- The kernel caught a signal while it was executing the call.

See Also

libc, mkfs, statfs()

Note

ustat() is largely superceded by **statfs()**.

utime() — System Call (libc)

Change file access and modification times
#include <sys/types.h>
#include <utime.h>
int utime(*file, times***)**
char ****file*;**
time_t *times***[2];**

utime() sets the access and modification times associated with the given *file* to times obtained from *times***[0]** and **times[1]**, respectively. The time of last change to the attributes is set to the time of the **utime()** call.

This call must be made by the owner of *file* or by the superuser.

Files

<sys/types.h>

See Also

libc, restor, stat(), utime.h
POSIX Standard, §5.6.6

Diagnostics

utime() returns -1 on errors, such as if *file* does not exist or the invoker not the owner.

utime.h — Header File

Declare system call utime()

The header file **<utime.h>** declares the COHERENT system call **utime()**. It also defines the structure **utimbuf**, which **utime()** uses.

See Also

header files, utime()

utmp — System Administration

File that notes login events that are active
/etc/utmp

File **/etc/utmp** notes every login event that is active — that is, when the user has logged in and has not yet logged out. It is read by the command **who** to display the users who are now logged into your system.

utmp records each active login event as a record of type **utmp**, which is defined in header file **<utmp>**. For details, see the Lexicon entry **utmp.h**.

File **/usr/adm/wtmp** records every login event that has concluded. You can comb this file to trace which user have logged onto your system, and when.

See Also

Adminstering COHERENT, utmp.h, wtmp

utmp.h — Header File

Login accounting information
#include <utmp.h>

Header file **<utmp.h>** defines the types and constants that are used to manipulate the system-adminstration files **/etc/utmp** and **/usr/adm/wtmp**. The former file describes every user who is currently logged into your system; the latter records when each user logged into your system and logged out again.

Each of these files consists of records, each of which has are objects of type **utmp**, which **<utmp.h>** defines as follows:

```
struct      utmp {
      char ut_user[8];
      char ut_id[4];
      char ut_line[12];
      short ut_pid;
      short ut_type;
      struct exit_status {
            short e_termination;
            short e_exit;
      } ut_exit;
      time_t ut_time;
};
```

The following describes each field in **utmp**:

ut_user
The login identifier of the user.

ut_id The user's identifier, as taken from **/etc/init**.

ut_line The device through which the user logged in.

ut_pid The process identifier of the user's shell.

ut_type
Type of entry in this file. This can be any of the following values:

EMPTY	An empty entry
RUN_LVL	Run level
BOOT_TIME	Boot time
OLD_TIME	
NEW_TIME	
INIT_PROCESS	Process spawned by **init**
LOGIN_PROCESS	A **getty** waiting for a login
USER_PROCESS	A user process
DEAD_PROCESS	
ACCOUNTING	

ut_exit The process's exit status. It consists of the following fields:

e_termination
Process's termination status.

e_exit Process's exit status.

ut_time
The time the user logged on.

The following functions use this header file:

endutent()	Close the logging file.
getutent()	Read the next entry from **/etc/utmp**.
getutid()	Find an entry in **/etc/utmp** by login identifier.
getutline()	Find an entry in **/etc/utmp** by login device.
pututline()	Write a record into **/etc/utmp**.
setutent()	Rewind the input stream that is reading **/etc/utmp**
utmpname()	Manipulate a file other than **/etc/utmp**.

Each function is described in its own Lexicon entry.

Files

/etc/utmp
/usr/adm/wtmp

See Also

ac, header files, login, utmp, who, wtmp

utmpname() — General Function (libc)

Manipulate a login logging file other than /etc/utmp
#include <utmp.h>
int utmpname(*file*)
**const char ****file*;

The system files **/etc/utmp** and **/usr/adm/wtmp** record information about every login event on your system — that is, they record every time someone logs into your system, the line from which the user logged in, and how long he was logged in. COHERENT comes with a set of functions that manipulate these files: they let you open a logging file, reads records, and update them.

By default, these functions manipulate the file **/etc/utmp**, which records the login events that are active — that is, the user has logged in but not yet logged out. Function **utmpname()** lets you change the file being manipulated. *file* points to the name of the file you wish to manipulate. Usually, this is the file **/usr/adm/wtmp**, which records login events that have concluded; but you can name any file in which you or the system has recorded login events. **utmpname()** also closes the logging file that is already open.

See Also

libc, utmp.h

utsname.h — Header File

Define utsname structure
#include <sys/utsname.h>

utsname.h defines the structure **utsname**. This structure holds information that describes a given release of the COHERENT system.

See Also

header files, uname()
POSIX Standard, §4.4.1

uuchk — Command

Check UUCP configuation
/usr/lib/uucp/uuchk [-I*file*] [v] [--help]

The command **uuchk** reads the UUCP configuration files **sys**, **port**, and **dial**, and generates a report on the configuration for each remote system listed in **sys**. You can use this report to repair problems in your configuration files.

The following gives sample output for system **mwcbbs**:

```
Call out using port intel.slow at speed 2400
The possible ports are:
 Port name intel.slow
  Port type modem
  Device /dev/com3fl
  Speed 2400
  Carrier available
  Hardware flow control available
  Dialer intel.slow
   Chat script "" AT\s&C1\s&D2\sE1\sM1\sQ0\sS0=0\sV1\sDP\D CONNECT\s2400
   Chat script timeout 60
   Chat failure strings BUSY NO\sCARRIER NO\sANSWER
   Chat script incoming bytes stripped to seven bits
   Wait for dialtone ,
   Pause while dialing ,
   Carrier available
   Wait 60 seconds for carrier
   When complete chat script "" \d+++\dAT\sH0\sE0\sV0\sQ1\sM0\sS0=1
   When complete chat script timeout 60
   When complete chat script incoming bytes stripped to seven bits
   When aborting chat script "" \d+++\dAT\sH0\sE0\sV0\sQ1\sM0\sS0=1
   When aborting chat script timeout 60
   When aborting chat script incoming bytes stripped to seven bits
Phone number 17085590445
Chat script "" \r\d\r in:--in: nuucp word: public word: 127417124
Chat script timeout 10
Chat script incoming bytes stripped to seven bits
At any time may call if any work
May make local requests when calling
May make local requests when called
May send by local request: /
May send by remote request: /usr/spool/uucppublic /tmp
May accept by local request: ~
May receive by remote request: /usr/spool/uucppublic /tmp
May execute rmail uucp
Execution path /bin /usr/bin /usr/local/bin
Will leave 50000 bytes available
Public directory is /usr/spool/uucppublic
Will use protocols g
For protocol g will use the following parameters
 window 3
 packet-size 64
```

uuchk recognizes the following command-line options:

-I*file*
--config*file*
Use *file* instead of the standard configuration files. This option lets you sanity-check a new configuration file without having to install it.

-v
--version
Print the version of **uuchk** and exit.

--help Print a help message, and exit.

See Also

commands, dial, port, sys, UUCP

uucico — Command

Communicate with a remote site
/usr/lib/uucp/uucico [-D] [-c*site***] [-I***file***] [-p***port***] [-r0] [-r1] [-s***site***] [-S***site***] [-x***level***]**

The UUCP daemon **uucico** is the program that communicates with a remote *site*. It either contacts another site and issues commands for execution by another **uucico** process on that remote system (*master* mode); or it receives a call from a remote system and executes the commands that that system issues (*slave* mode).

The commands **uucp** and **uux** invoke **uucico** automatically, usually in master mode. **uucico** can also be invoked directly from the shell, from within a script, or from with a **cron** file.

You can also name **uucico** in file **/etc/passwd** as the default process to run for a given login identifier. A system that logs in under that login ID (presumably, a version of **uucico** on a remote system) will interact with your system's **uucico**, instead of a shell. When invoked in this manner, **uucico** runs in slave mode by default.

After **uucico** has finished communicating with the remote system, it invokes the daemon **uuxqt** to execute the commands issued by the remote system. For information on **uuxqt**, see its Lexicon entry.

uucico recognizes the following command-line options:

-c*site* "Cron" mode: If a call is not permitted to *site* at the present time, then do not make the call; but also, do not log an error message or update the system status. Use this option if you wish to invoke **uucico** regularly through **cron**, and do not want to be bombarded with error messages should the entry in **cron** conflict with the legal calling times set in **sys**.

-D Do not detach from the device until the contact with the remote system concludes.

-e Force **uucico** to produce its own **login:** and **Password:** prompts. **uucico** checks the password it receives against its own, private list, rather than against the password kept in file **/etc/passwd**. This should be used with the options **-l** and **-p**. When used with this option, **uucico** does not terminate, but continues to issue prompts until you kill it explicitly. This option permits you to use **uucico** as a server on a network.

-f*site* Force option: call *site* immediately, regardless of whether the site's description in **sys** indicates that this is a legal time to call.

-I *file* Read configuration information from *file*, instead of from the default file **/usr/lib/uucp/sys**.

-l Force **uucico** to produce its own **login:** prompt. **uucico** checks the login it receives against its own, private list, rather than against the normal system password files. This should be used with the option **-e**.

-q Quiet: do not invoke daemon **uuxqt** on the remote system.

-p*port* Use *port*. When used with the options **-s** or **-S**, dial out on *port*; this overrides the default port used with the system being contacted. When **uucico** is in slave mode, this implies the option **-e**.

-r0 Act as slave in polling process; that is, carry out the orders of another **uucico** that has dialed into your system. This is the default.

-r1 Act as master in polling process; that is, dial out to another system and give it orders. This option is implied by options **-s** or **-S**. If the **uucico** command line does not name a site to call, this option tells **uucico** to call any system for which work is waiting to be performed.

-s*site* Call *site*. This must name one of the entries in **/usr/lib/uucp/sys**.

-S*site* Call *site* immediately, if the present time lies within the legal time set for *site*, as described in file **/usr/lib/uucp/sys**.

-w After contacting a system with the options **-r1**, **-s**, **-S**, begin an endless loop of login prompts, as with the option **-e**. In effect, UUCP calls a remote site; but instead of logging into that site, it lets that site log into it.

-x*activity***[,***activity***,...,***activity*^**]**
-X*activity***[,***activity***,...,***activity***]**
Log a given *activity*. These logs can help you debug problems with UUCP. **uucico** recognizes the following activities:

abnormal	**chat**	**config**
execute	**handshake**	**incoming**
outgoing	**port**	**proto**
spooldir	**uucp-proto**	

One **-x** option can name multiple activities, separated by commas. A **uucico** command line can contain more than **-x** option. **uucico** writes its logging information into file **/usr/spool/uucp/.Admin/audit.local**.

Example

To poll the site **mwcbbs** (the Mark Williams bulletin board) five minutes after each hour, put the following entry into a **cron** file:

```
05 * * * * /usr/lib/uucp/uucico -smwcbbs -r1
```

Files

/usr/lib/uucp/sys — List of reachable systems
/usr/spool/uucp/.Log/uucico/*sitename*— **uucico** activities log file for *sitename*
/usr/spool/uucp/.Log/uucico/UUCICO— uucico debug log
/usr/spool/uucp/*sitename*— Spool directory for work

See Also

commands, cron, uucp, UUCP, uulog, uutouch, uuxqt

Notes

uucico was written by Ian Lance Taylor (ian@airs.com).

uuconv — Command

Convert UUCP configuration files to Taylor format
/usr/lib/uucp/uuconv -i *input* **-o** *output* **[-p** *program***] [-I** *file***]**

The command **uuconv** converts UUCP configuration files from one format to another. In all probability, you will have to run this program only once, when you convert from your previous UUCP implementation to Taylor UUCP.

uuconv recognizes the follow command-line options:

--help Print a help message and exit.

-I *file*
--config *file*
Read configuration information from *file* instead of from the standard UUCP configuration file.

-i *file*
--input *file*
Read input from *file*.

-o *file*
--output *file*
Write output into *file*.

-p *program*
--program *program*
Convert *program* (e.g., **uucp** or **cu**).

-v
--version
Print the version of **uuconv** that you are running, and exit.

See Also

commands, UUCP

Notes

uuconv was written by Ian Lance Taylor (ian@airs.com).

UUCP — Overview

Unattended communication with remote systems

UUCP stands for “UNIX to UNIX communications protocol”. It is a system of commands that allows you to exchange files with other COHERENT or UNIX systems, in an unattended manner. With UUCP, you can send mail to other systems, upload files, and execute commands. When configured correctly, UUCP also lets other users upload files to your system, copy files from it, and execute commands. All this can be done without your having to sit at your console and type commands; thus, files can be transferred in the small hours, when telephone rates are lower and computers are relatively free.

UUCP gives you access to the Usenet, a nation-wide network of UNIX and COHERENT users. Access to the Usenet will let you exchange mail with any of the thousands of Usenet users, receive mail from them, download source code for many useful programs, and read the latest news on a host of subjects. For details on contacting UUNET, a commercially accessible Usenet site, enter the command:

```
phone uunet
```

Implementation of UUCP

Beginning with release 4.2, COHERENT implements the Taylor UUCP package. The current implementation is Taylor UUCP version 1.05. Taylor UUCP offers extraordinary flexibility, beyond that offered by standard implementations of UUCP. The following Lexicon entries describe UUCP:

config.	Overall configuration file for UUCP
cu	Introduce the **cu** communications utility
dial	Describe how **uucico** and **cu** can dial a modem
domain	Describe the file that names your UUCP domain
port	File that describes ports through which UUCP dials

sys	File that describes systems contacted by UUCP
uuchk	Check UUCP configuation
uucico	Daemon that controls communication with a remote site
uuconv	Convert UUCP configuration files to Taylor format
uucp	Spool files for transmission to other systems
uucpname	File that names your system
uudecode	Decode a binary file sent from a remote system
uuencode	Encode a binary file for transmission
uuinstall	Install or modify UUCP
uulog	Read a UUCP log
uumkdir	Create a UUCP directory
uumvlog	Archive UUCP log files
uuname	List UUCP names of known systems
uupick	Pick up a file uploaded from a remote system
uurmlock	Remove a UUCP lockfile
uusched	Call all systems that have jobs waiting for them
uustat	Display and modify the status of a UUCP job
uuto	Send a file to a remote system
uutouch	Touch a file to trigger **uucico** poll
uutry	Debugging tool for UUCP
uux	Execute a command on a remote system
uuxqt	Execute commands requested by a remote system

Files and Directories

UUCP uses the following files and directories:

/usr/lib/uucp/sys
This file contains information about remote UUCP sites with which you can communicate. **uucico** uses its information to connect to remote systems; sets permissions for the directories that a given remote system can write into or read from; establishes the protocol (or protocols) that can be used when communicating with the given remote system to transfer files.

/usr/lib/uucp
This directory holds many of the UUCP executables. It also holds the following configuration files:

/usr/lib/uucp/config
Customize the configuratio of Taylor UUCP. Note that this file is not shipped with COHERENT, to ensure that the default configuration is used; however, you can write one yourself easily enough. For details, see the Lexicon entry **config**.

/usr/lib/uucp/dial
uucico uses the information in this file to communicate with modems.

/usr/lib/uucp/port
uucico uses the information in this file to communicate with a given port on your system.

/usr/spool/uucp
This directory holds log files and spool directories, as follows:

/usr/spool/uucp/.Admin
This directory holds the following administrative logging files:

/usr/spool/uucp/.Admin/xferstats
This file holds statistics about the rate at which data were transferred between your site and a remote site.

/usr/spool/uucp/.Admin/audit.local
This file holds auditing information, as generated using the option **-x** with any UUCP command.

/usr/lib/uucp/.Log
This directory holds information that detail the files transferred between your system and any remote system. It contains one sub-directory for each UUCP command — one each for **uucico**, **uucp**, **uux**, and **uuxqt**. Each sub-directory, in turn, contains one log file for each remote system with which your system exchanges files, plus the file **ANY**, which holds information about all remote systems. For example, file **/usr/spool/uucp/.Log/uucp/lepanto** logs every file that you

have exchange with remote site **lepanto** via the command **uucp**.

/usr/spool/uucp/.Received
This directory contains one sub-directory for each remote system with which your system exchanges files. It holds files received from that system that cannot be executed properly. If your system is configured correctly, this directory should be empty.

/usr/spool/uucp/.Sequence
This directory holds one file for each remote system with which you exchange files. The file holds a string from which the job most recently performed with that site was named. This sequence number is used to identify each job uniquely. This is discussed in more detail below.

/usr/spool/uucp/.Status
This directory holds one file for each system with your system communicates via UUCP. The file holds information about the status with which the last contact exited. For example, if your system communicated successfully with system **mwc**, then file **/usr/spool/uucp/.Status/mwc** will hold an entry that resembles the following:

```
0 0 778536664 0 SUCCESSFUL mwc
```

However, if your system communicates with system **sales** and the last session failed during handshake, then file **/usr/spool/uucp/.Status/sales** will hold something like the following:

```
4 7 769981110 4200 Handshake failed sales
```

Note that if a **.Status** file indicates that the last contact failed, **uucico** may silently refuse to dial out to that system; UUCP is designed this way, in order to spare you the expense of repeatedly calling a system whose connection is damaged in some way. The solution is simply to remove the file in question. For example, if **uucico** refuses to dial system **mwc** and you know that that system is working correctly, try removing file **/usr/spool/uucp/.Status/mwc**.

/usr/spool/uucp/.Temp
This directory holds one directory for each system with which your system has exchanged files. Each sub-directory holds temporary files used by the jobs being performed for that system.

/usr/spool/uucp/.Xqtdir
The command **uuxqt** executes from within this directory all commands that have been spooled onto your system for execution. It also copies into this directory all files on remote systems that a spooled command names. Note that files reside here only briefly.

/usr/spool/uucp/*sitename*
This directory holds all files being uploaded to site *sitename*. Each file is constructed as follows:

prefix This is either **D.** or **C.**. The former indicates a data file, and the latter a command file (that is, a file to be executed on the remote system by command **uux**).

site The name of the site to which the file is being uploaded.

sequence_number
This is a unique number, meant to ensure that no UUCP file clobbers another. When UUCP is spooling a file to be transmitted to a remote site, it looks in that site's **.Sequence** file, increases the sequence number by one, uses that number to name the file, and writes the incremented sequence number back into the site's **.Sequence** file.

/usr/spool/uucp/LCK..*
Finally, files that begin with the string **LCK..** are lock files. UUCP (and many other COHERENT programs) use them to lock devices, to ensure that only one program can access a device at a time. Each lock file contains the process identifier of the process that has locked that device, but different programs use different conventions in naming lock files.

Programs that log users into your system lock console and terminal devices. These programs use lock files whose names are built from the major-device number and the minor-device number of the device being locked For example, file **/usr/spool/uucppublic/LCK..2.1** locks the device with major number 2 and minor number 1 — that is, the color virtual-console device **/dev/color1**. Looking into file **LCK..2.1**, we see the number 6836; and when we use the command **ps -alx** to look for a process with this identifier, we see the following

```
color1   6836     1     fred  133 6001 w           ksh
color1   8923  6836     fred  204 6001 S  ttywait  me
```

That is, user **fred** has logged into this system via device **/dev/color1** and invoked a shell that has process identifier **6836**.

Second, when UUCP opens a port to dial out, it creates a lock file whose name includes the name of the port on which it is dialing. For example, if UUCP is dialing out via port **/dev/com3fl**, it creates file **LCK..com3fl** in **/usr/spool/uucp**. This helps to stop two UUCP process from each trying to open the same port at the same time.

Finally, when UUCP dials a given remote site, it creates a lock file for that site. For example, if UUCP dials site **mwc**, it creates lock file **LCK..mwc** in directory **/usr/spool/uucp**. This help to prevent two different UUCP processes from attempting to dial the same site at the same time.

This concludes our discussion of UUCP's files and directories. For more information, see the Lexicon entries **config**, **dial**, **port**, and **sys**.

Permissions

The following gives the correct permissions and ownership for the files that comprise the UUCP system:

```
-rw-------  uucp   uucp  /usr/lib/uucp/dial
-rw-------  uucp   uucp  /usr/lib/uucp/port
-rw-------  uucp   uucp  /usr/lib/uucp/sys
-r-sr-xr-x  uucp   root  /usr/lib/uucp/uucico
-rwxr-xr-x  uucp   root  /usr/lib/uucp/uuconv
-r-s--s--x  root   root  /usr/lib/uucp/uumkdir
-r-xr-xr-x  uucp   uucp  /usr/lib/uucp/uumvlog
-r-xr-xr-x  uucp   uucp  /usr/lib/uucp/uurmlock
-r-xr-xr-x  root   root  /usr/lib/uucp/uusched
-r-s--s--x  uucp   uucp  /usr/lib/uucp/uutouch
-r-x------  uucp   uucp  /usr/lib/uucp/uutry
-r-sr-xr-x  uucp   root  /usr/lib/uucp/uuxqt
-r-s--s--x  uucp   uucp  /usr/bin/uucheck
-r-sr-xr-x  uucp   root  /usr/bin/uucp
-r-x--x--x  bin    bin   /usr/bin/uudecode
-r-x--x--x  bin    bin   /usr/bin/uuencode
-r-s--s---  uucp   uucp  /usr/bin/uuinstall
-rwxr-xr-x  root   root  /usr/bin/uulog
-r-sr-xr-x  uucp   root  /usr/bin/uuname
-rwxr-xr-x  root   root  /usr/bin/uupick
-r-sr-xr-x  uucp   root  /usr/bin/uustat
-r-xr-xr-x  root   root  /usr/bin/uuto
-r-sr-xr-x  uucp   root  /usr/bin/uux
```

Permissions should be set properly by COHERENT when you installed it on your computer. However, if problems arise with UUCP, be sure to check that permissions are correct. If permissions have somehow been reset incorrectly, UUCP will not work because much of its work depends upon its being able to create and delete files in certain restricted directories.

Should a file's permissions be "stepped on" for whatever reason, use the command **chmod** to restore them. Likewise, should the group or user who "owns" a file or directory be changed for whatever reason, you (or, to be more exact, the superuser **root** can use the commands **chgrp** and **chown** to restore proper ownership. For details on how to use these commands, see their entry in the Lexicon.

Debugging UUCP Problems

For information how to debug and solve common problems with UUCP, see the tutorial on UUCP that appears in the front half of this manual.

See Also

asy, commands, config, cu, dial, domain, modem, mwcbbs, port, sys, terminal, uuchk, uucico, uuconv, uucp, uucpname, uudecode, uuencode, uuinstall, uulog, uumkdir, uumvlog, uuname, uupick, uurmlock, uusched, uustat, uuto, uutouch, uutry, uux, uuxqt
UUCP, Remote Communications Utility, tutorial

Notes

The Lexicon entry **mail** gives directions on how to send mail to users on popular commercial networks.

For information on how to hook up a Trailblazer modem to run UUCP, see the Lexicon entry for **modem**.

The COHERENT implementation of UUCP was written by Ian Lance Taylor (ian@airs.com). It was ported to COHERENT by Robert Chalmers (earth@nanguo.cstpl.com.au). For information on copyright and availablity of source code, see the documentation included in file **/usr/src/alien/uudoc.tar.Z**.

uucp — Command

Spool files for transmission to other systems
uucp [-cCdfmr] [-n*user***] [-x***activity***]** *source ... dest*

The command **uucp** spools every file *source* for copying to *dest*. *source* and *dest* can specify a remote system.

uucp recognizes the following options:

-C Copy the source file into spool directory; same as option **-p**. This is the default.

-c Do not copy the source file into spool directory; rather, use the file itself. The file must be readable both by yourself and by the daemon **uucico**. If the file is removed before **uucico** processes it, the transmission of the file will fail.

-d Create directories as required on the destination system. This is the default.

-f Do not create any directories on the remote system. If directories do not already exist, abort copying the file.

-g*grade*
Assign a grade (a single ASCII character, from ‘0’ through ‘z’) to indicate the importance of the file being transmitted. The lower the ASCII value of *grade*, the more important the file; thus, ‘0’ is the highest grade and ‘z’ the lowest.

-I *file*
Read the configuration for the remote system from *file* instead of from **/usr/lib/uucp/sys**, which is the default.

-j Report the job’s process identifier. If you wish, you can use this identifier with the command **uustat** to kill the job.

-m Send mail to requester when the file is sent; report whether the job was executed successfully.

-n*user*
Send mail to *user* on destination system when the file is received. *user* can contain a path. Note that *user* is relative to the destination machine, not to originating machine or to any intervening machine. For example, consider the command:

```
uucp -nlepanto!fred myfile joe!/tmp
```

Here, you are copying **myfile** from your machine into directory **/tmp** on machine **joe**, and sending notification to user **fred** on machine **lepanto**. If, however, machine **joe** does not know how to address machine **lepanto**, then **fred** will never be notified of the transfer.

-p Copy the source file into spool directory; same as **-C**. This is the default.

-R Copy directories recursively.

-r Spool transfer request, but do not initiate **uucico**.

-s *file*
Write status upon completion of job into *file*.

-u *user*
Set user name to *user*.

-W Do not prefix the file’s name with the name of the current directory.

-x*activity*
Log a given *activity*. These logs can help you debug problems with UUCP. **uucp** recognizes the following activities:

abnormal
Log abnormal events that occur while spooling a file for copying.

config Log problems that arise with reading or interpreting the configuration files **dial**, **port**, and **sys**.

execute
Log each step **uucp** takes as it executes.

spooldir
Log activity that involved the UUCP spooling directory **/usr/spool/uucp** and its subdirectories.

uucp writes its logging information into file **/usr/spool/uucp/.Admin/audit.local**.

Examples

The first example copies file **foo** to directory **/bar** on system **george**:

```
uucp foo george!/bar
```

The next example copies file **/foo** from system **george** into directory **/tmp** on your system:

```
uucp george!/foo /tmp
```

The next example copies file **/foo** from system **george** into file or directory **/bar** on system **ivan**:

```
uucp george!/foo ivan!/bar
```

Note that this assumes your system can talk to both **george** and **ivan** and that your system has permission to read file **/foo** on system **george** as well as to write file **/bar** on system **ivan**.

The next example downloads files **/foo** and **/bar** from remote systems **ivan** and **george** into directory **/tmp** on your system:

```
uucp ivan!/foo george!/bar /tmp
```

The last example downloads file **foo** from system **ivan** via system **george**:

```
uucp george!ivan!foo
```

For an example of using the command **find** with **uucp** to spool files automatically, see the entry for **find**.

Files

/usr/lib/uucp/sys — List of reachable systems
/usr/spool/uucp/.Log/*/*sitename*— **uucp** activities log files for *sitename*
/usr/spool/uucp/*sitename*— Spool directory for work

See Also

commands, mail, uucico, UUCP, uudecode, uuencode, uutouch, uuxqt

Notes

uucp was written by Ian Lance Taylor (ian@airs.com).

uucpname — System Administration

Set the system's UUCP name
/etc/uucpname

The file **/etc/uucpname** sets the name by which your system is known to all other system with which it communicates via UUCP. To rename your system, simply change the contents of this file.

The contents of **/etc/uucpname** is, in effect, your system's *nom de plume*. It should be unique (or as unique as possible), easily remembered, and preferably in good taste. Examples of existing systems include **lepanto**, **smiles**, and **stevesf**. You should avoid names taken from popular culture, such as **calvin**, **hobbes**, or **terminator**: many other people have already used them.

Note that system names must obey the following rules:

- UUCP names must be no more than 14 characters long.
- Names must consist of letters and numbers. No punctuation marks, white space, control characters, or diacritic marks are permitted.

- Each name must begin with a letter.

If you wish for your system to communicate with other systems in the world-wide UUCP network, you should follow the following restrictions as well:

- UUCP names should be contain no more than seven characters.
- They should use only lower-case letters and digits.

If you are connecting to other machines we recommend that you acquire a registered Fully Qualified Domain Name. Every person in the United States may register in the **.us** domain. Send mail to **us-domain-request@venera.isi.edu** for information on this. If you wish to create your own domain (e.g., **mwc.com)**, send mail to **info-request@uunet.uu.net** for information on this.

See Also

Administering COHERENT, domain, UUCP

Notes

Only the superuser **root** can edit **/etc/uucpname**.

uudecode — Command

Decode a binary file sent from a remote system
uudecode [*file* **]**

uudecode takes a **file** encoded by **uuencode** and translates it back to binary. Any leading and trailing lines added by **uucp** are discarded.

If the *file* is not specified, standard input is read.

Example

Consider the file **tmp** consisting of:

```
begin 644 sys
M5&AE(’%U:6-K(&)R;W=N(&9O>"!J=6UP<R!O=F5R(’1H92!L87IY(&1O9RX*

end
```

Note that the third line is a space followed by a newline. To decode it, type:

```
uudecode tmp
```

The output contained in file **sys** will be:

```
The quick brown fox jumps over the lazy dog.
```

See Also

commands, UUCP, uucp, uuencode

Notes

The user on the remote system must be able to write the file.

uuencode — Command

Encode a binary file for transmission
uuencode [*source* **]** *file_label* **[<** *source* **] >** *output*

uuencode prepares a file for transmission to a remote destination via **uucp.** It takes binary input and produces an encoded version, consisting of printable ASCII characters, on standard output, which may be redirected or piped to **uucp**.

If *source* is not specified, **uuencode** reads the standard input and writes to the standard output. If, however, *source* is specified, **uuencode** its permissions into the **uuencode**’d file. *file_label* is the name that **uudecode** gives to the file when it is decoded.

uuencode is chiefly used for mail. You cannot mail a binary file, but you can mail a **uuencode**’d binary. The standard way to mail a binary is to compress it, **uunecode** it, split it into pieces less than 50 kilobytes each, then mail each piece.

The format of the encoded file is as follows:

1. A *header* line starting with the characters **begin** followed by a space. This is followed by the mode of the file in octal and the name of the output file specified on the command line. (For details, see the Lexicon entry for **chmod**). These last two fields are also separated by a space. The mode and the system name can be changed by directing the output into a file and editing it.
2. The *body* of the file, consisting of a number of lines, each no more than 62 characters long, including a newline character. Each line starts with a character count written as a single ASCII character, representing an integer value from 0 (octal 40) to 63 (octal 135) giving the number of characters in the rest of the line. This is followed by the encoded characters and a newline. The last line of the body is a line consisting of an ASCII space (octal 40).
3. A *trailer*, which consists of the string **end** on a line by itself.

The encoding is done by taking three bytes and storing them in four characters, six bits per character. Each six bits is converted to a printable character by adding 0x20 to it.

Example

Consider the file **tmp**, which consists of the line:

```
The quick brown fox jumps over the lazy dog.
```

To record it in file **tmp.send**, type:

```
uuencode tmp < tmp > tmp.send
```

The output is:

```
begin 644 tmp
M5&AE('%U:6-K(&)R;W=N(&9O>"!J=6UP<R!O=F5R('1H92!L87IY(&1O9RX*
 
end
```

Note that the third line consists of a space followed by a newline.

See Also

commands, UUCP, uucp, uudecode

Notes

uuencode expands a file by more than one third, which thus increases transmission time. This can be a factor when sending large files.

uuinstall — Command

Install or modify UUCP
uuinstall

uuinstall help you to install UUCP on your COHERENT system. It uses screen templates, help lines, and prompts to walk you through the installation of devices, remote systems, site names, domains, and permissions. For a detailed description of its use, see the tutorial on UUCP in the front of this manual.

See Also

commands, UUCP

Notes

Only the superuser **root** can execute **uuinstall**.

On some terminals, the arrow keys do not move the cursor. In this case, you can use **vi**-style cursor-movement keys:

H	Move the cursor left
K	Move the cursor up
L	Move the cursor right
J	Move the cursor down

uulog — Command

Read a UUCP log
uulog [-f*system***] [-s***system***] [-n***number***] [-x**]

uulog copies the last part of the file **/usr/spool/uucp/.Log/uucico/***system* to see what **uucico** has done recently. *system* names the remote system whose logfile will be examined. If it is not specified, logfiles for all systems are displayed.

uulog recognizes the following options:

-f*system*
Similar to the command **tail -f**: this forces **uulog** to display UUCP activity as it is written into the log file for *system*, until you interrupt it by typing **<ctrl-C>**.

-n*number*
Display only *number* lines from the end of the log.

-s*system*
Display the log file for *system*.

-x Display the log file for the command **uuxqt** rather than **uucico**.

Files

/usr/spool/uucp/.Log/uucico/*system*— **uucico** log file for *system*
/usr/spool/uucp/.Log/uuxqt/*system*— **uuxqt** log file for *system*

See Also

commands, UUCP

uumkdir — Command

Create UUCP directories
/usr/lib/uucp/uumkdir [-m *mode* **] [-p]** *directory ...*

The command **uumkdir** creates each *directory* named on its command line. Option **-m** sets the permissions on the newly created directory to *mode*, which must be a three-numeral octal number.

See Also

commands, UUCP

uumvlog — Command

Archive UUCP log files
uumvlog *days*

uumvlog copies all UUCP log files into backup files, named for their respective commands and the date upon which the backup was performed. *days* gives the number of days for which backup files should be kept: if a backup file is more than *days* days old, then **uumvlog** will delete it.

This command should be run by **cron**, because the UUCP log files can threaten to exhaust available file space on a small system unless they are chopped back daily.

Files

/usr/spool/uucp/.Log/*command***/***system*— UUCP log files

See Also

commands, cron, UUCP

Notes

uumvlog manages the log files under directory **/usr/spool/uucp/.Log**. However, it does not touch the files in **/usr/spool/uucp/.Admin**. These can grow quite large if unattended. At present, you must manage these files by hand.

uuname — Command

List UUCP names of known systems
uuname [-l]

The command **uuname** lists the names of all systems reachable directly by UUCP. It does so by reading the names of the systems defined in file **/usr/lib/uucp/sys**, plus the name of your local system as set in file **/etc/uucpname**. Command-line option **-l** prints the name of your local system only.

Files

/etc/uucpname — Name of local system
/usr/lib/uucp/sys — Site and remote login data

See Also

commands, UUCP

uupick — Command

Pick up a file uploaded from a remote system
/usr/bin/uupick [-s *system***] [-I** *file***] [-x** *event***]** *file ...*

The command **uupick** lets you "pick up" each *file* that has been uploaded to your system via UUCP. It moves the file into your current directory from whence it was copied on your system. It usually used to acquire files that had been sent to your system via the script **uuto**.

uupick recognizes the following command-line options:

--help Print a help message, and exit.

-I *file*
--config *file*
Read configuration information from *file* instead of from the default configuration file.

-s *system*
--system *system*
"Pick up" only files uploaded from *system*.

-v
--version
Print the version of **uupick** that you are running, and exit.

-x *activity*

-x*type***[,***type***,...,type]**
-X*type***[,***type***,...,type]**
Log a given *activity*. **uupick** recognizes the following activities:

abnormal	**chat**	**config**
execute	**handshake**	**incoming**
outgoing	**port**	**proto**
spooldir	**uucp-proto**	

One **-x** option can name multiple activities, separated by commas. A **uupick** command line can contain more than **-x** option.

See Also

commands, UUCP, uuto

Notes

uupick was written by Ian Taylor (ian@airs.com).

uurmlock — Command

Remove UUCP lock files
uurmlock

UUCP uses a system of lock files to ensure that sites are polled in an orderly manner. It creates a lock file named after the site being polled, to prevent more than one invocation of **uucico** or another UUCP command from polling the same site at the same time. On occasion, if UUCP fails or crashes, it will neglect to clean up its lock files, thus preventing itself from polling the locked sites.

The command **uurmlock** removes all lock files. You should run this if you suspect that UUCP has died in a disorderly manner and has left lock files lying around unattended.

Before you run **uurmlock**, examine the output of the command **ps** to ensure that no UUCP command is running at the moment (and so has legitimately locked a site).

Files

/usr/spool/uucp/LCK.* — UUCP lock files

See Also

commands, UUCP

Notes

Only the superuser **root** can run **uurmlock**.

Note that **uurmlock** removes all **.LCK** files from **/usr/spool/uucp**. Not all of these are used by UUCP; however, this behavior is necessary to remain compatible with UNIX, and is almost always benign.

uusched — Command

Call all systems that have jobs waiting for them
/usr/lib/uucp/uusched

The one-line script **uusched** invokes command **uucico** with its option **-r1**, which tells **uucico** to call all systems that have jobs queued up for them.

See Also

commands, uucico, UUCP

uustat — Command

UUCP status inquiry and control
uustat [-eKiMNQ] [-B *lines***] [-cC** *command***] [-o** *hours***] [-sS** *system***] [-uU** *user***] [-y** *hours***]**
uustat -a
uustat [-k *jobid***] [-r** *jobid***]**
uustat -m
uustat -p
uustat -q

The command **uustat** displays status information about the UUCP system. You can also use it to cancel or rejuvinate requests made by via commands **uucp** or **uux**.

By default, **uustat** displays every job queued by the user who invokes this command. If the command line includes any of the options **-a**, **-e**, **-s**, **-S**, **-u**, **-U**, **-c**, **-C**, **-o**, or **-y**, then **uustat** displays information about all of the jobs that match given specifications.

The option **-K** can be used to kill a selected group of jobs, such as all jobs more than seven days old.

Command-line Options

uustat recognizes the following command-line options:

-a List all queued requests to transfer files.

-C *command*
List all jobs except those that request the execution of *command*. If *command* is **ALL**, list all jobs that simply request a file transfer (as opposed to requesting the execution of some command). You cannot use this option with the option **-c**. A **uustat** command can hold more than one **-C** option.

-c *command*
List every job that requests the execution of *command*. If *command* is **ALL**, **uustat** lists all jobs that request the execution of a command (as opposed to simply requesting a file transfer). A **uustat** command can hold more than one **-c** option.

-e List queued requests to execute a program on a remote system, rather than queued requests to transfer files. Queued execution requests are processed by **uuxqt** rather than **uucico**. A queued execution request may await a file from a remote system. These requests are created by an invocation of the command **uux**.

-I *file* Read configuration information from *file* instead of from the default file **/usr/lib/uucp/sys**.

-i For each listed job, prompt whether to kill the job. If the first character of the input line is **y** or **Y**, the job will be killed.

-K Kill each listed job without prompting for permission. This can be used in a script to clean up obsolete jobs automatically.

-k *jobid* Kill the job with the identifier *jobid*. A job's identifier is shown by the default output format, as well as by the commands **uucp** or **uux** when invoked with option **-j**. A job may only be killed only by the user who created the job, the UUCP administrator, or the superuser **root**. You can use the option **-k** more than once on a **uustat** command line, to kill several jobs simultaneously.

-M For each listed job, send mail to the UUCP administrator. If the job is killed (due to **-K** or **-i** with an affirmative response), the mail will indicate that. A comment specified by the **-W** option may be included. If the job is an execution, the initial portion of its standard input will be included in the mail message; the number of lines to include may be set with the **-B** option (the default is 100). If the standard input contains null characters, it is assumed to be a binary file and is not included.

-m Display the status of conversations for all remote systems.

-N For each listed job, send mail to the user who requested the job. The mail is identical to that sent by the option **-M**.

-o *hours*
List all jobs that have been queued longer than *hours*.

-p Display the status of all processes holding UUCP locks on systems or ports.

-Q Work quietly: Do not list the job, just perform the actions indicated by the options **-i**, **-K**, **-M**, or **-N**.

-q Display the status of commands, executions, and conversations for all remote systems for which commands or executions are queued.

-r *jobid* Rejuvinate the job with job identifier *jobid*. This marks the job as having been invoked at the current time; which, in turn, affects the output of the options **-o** or **-y** and preserves the job from any automated cleanup daemon. The job identifier is shown by the default output format, as well as by the commands **uucp** and **uux** when invoked with option **-j**. A job may only be rejuvenated by the user who created the job, by the UUCP administrator, or the superuser **root**. You can use the option **-r** more than once on a **uustat** command line, to rejuvinate several jobs simultaneously.

-S *system*
List all jobs except the ones queued for *system*. You cannot use this option with the option **-s**. A **uustat** command can hold more than one **-S** option.

-s *system*
List every job queued for *system*. A **uustat** command can hold more than one **-s** option.

-U *user* List all jobs except the ones queued for *user*. You cannot use this option with the option **-u**. A **uustat** command can hold more than one **-U** option.

-u *user* List every job queued for *user*. A **uustat** command can hold more than one **-u** option.

-W Specify a comment to be included in mail sent with the **-M** or **-N** options.

-x *type* Turn on particular debugging types. The following types are recognized:

abnormal	**chat**	**config**
execute	**handshake**	**incoming**
outgoing	**port**	**proto**
spooldir	**uucp-proto**	

Only **abnormal**, **config**, **spooldir**, and **execute** are meaningful for **uustat**.

Multiple types may be given, separated by commas, and the **-x** option can appear multiple times on the **uustat** command line. A number may also be given, which will turn on that many types from the foregoing list; for example, **-x 2** is equivalent to **-x abnormal,chat**.

-y *hours*
List all jobs that have been queued less than *hours*.

Examples

The first example displays the status of all jobs:

```
uustat -a
```

The output has the format:

jobid system user queue-date command (size)

The job identifier may be passed to the options **-k** or **-r**. The size indicates how much data is to be transferred to the remote system, and is absent for a file-receive request. The options **-s**, **-S**, **-u**, **-U**, **-c**, **-C**, **-o**, and **-y** may be used to control which jobs are listed.

The next example displays the status of queued execution requests:

```
uustat -e
```

The output has the format:

system requestor queue-date command

The options **-s, -S, -u, -U, -c, -C, -o,** and **-y** can be used to control which requests are listed.

The next example displays the status for all systems with queued commands:

```
uustat -q
```

This displays the system, the number of commands queued for it, the age of the oldest queued command, the number of queued local executions, the age of the oldest queued execution, the date of the last conversation, and the status of that conversation.

The next example displays conversation status for all remote systems:

```
uustat -m
```

The output gives the system, the date of the last conversation, and the status of that conversation. If the last conversation failed, **uustat** indicates how many attempts have been made to call the system. If the retry period is preventing calls to that system, **uustat** also displays the time when the next call will be permitted.

The next example displays the status of all processes that hold UUCP locks:

```
uustat -p
```

The output is exactly the same as that of the command **ps** for each process that holds a lock.

The next example kills all **rmail** commands that have been queued up waiting for delivery for over one week (168 hours).

```
uustat -c rmail -o 168 -K -Q -M -N -W"Queued for over 1 week"
```

uustat sends mail both to the UUCP administrator and to the user who requested the **rmail** execution. The mail message includes the string given by the **-W** option. The option **-Q** prevents any of the jobs from being listed on the terminal, so any output from the program will be error messages.

Files

/usr/lib/uucp/config — Configuration file.
/usr/spool/uucp — UUCP spool directory.

See Also

commands, ps, rmail, uucico, UUCP, uucp, uux, uuxqt

Notes

uustat was written by Ian Lance Taylor (ian@airs.com).

uuto — Command

Send a file to a remote system
/usr/bin/uuto *file ... file remote_system*

The one-line script **uuto** invokes the command **uucp** to send each *file* to *remote_system*.

See Also

commands, UUCP, uucp

uutouch — Command

Touch a file to trigger UUCP poll
uutouch *system*

The command **uutouch** creates an empty control file for *system* in the directory **/usr/spool/uucp/***system*. This forces UUCP to poll *system* when **uucico** is called with the option **-sall**. If the empty file for *system* aready exists, it is left alone.

There are three types of files in the spool directory **/usr/spool/uucp/***system*:

C. Command file.

D. Data file.

X. Execute file.

Example

A typical usage is to put the following line into the **cron** file **/usr/spool/cron/crontabs/uucp**:

```
0 7 * * * /usr/lib/uucp/uutouch george
```

This forces UUCP to schedule a poll to the remote system **george** at 7 AM local time. The actual poll take place when **uucico** is started.

Files

/usr/spool/uucp/*sitename* — Directory for uucp work files

See Also

commands, cron, uucico, UUCP

uutry — Command

Debugging script for UUCP
uutry *remotesystem* **[-x***debuglevel***]**"

The command **uutry** is a script that invokes **uucico** to contact *remotesystem*, and records all debugging information that **uucico** generates. **uutry** redirects the debugging information into file **audit.local** in directory **/usr/spool/uucp/.Admin**. If such a file already exists, **uutry** renames it **audit.OLD** before it invokes **uucico**.

The option argument **-x** sets the debugging level to *debuglevel*. This is a number from zero through nine; for information on what the debugging level means, see the Lexicon entry for the command **uucico**. The default level is five.

See Also

commands, UUCP

Notes

For security reasons **uutry** can be run only by the superuser **root**.

uux — Command

Execute a command on a remote system
uux [-a *user***] [-rnpz]** *command-string*

The command **uux** spools *command-string* for execution on a remote system. Usually, it is invoked by software systems, in particular the mail system, to request that work be performed on a remote system. However, you can also invoke **uux** by hand to execute a task on a remote system.

For security reasons, you can execute on the remote system only the commands that the remote system permits explicitly. These commands are named in the entry for your system in the remote system's copy of **/usr/lib/uucp/sys**.

If all permissions are in order, **uux** creates a file with the prefix **X.** in the remote system's directory **/usr/spool/uucp/***yoursystem*, where *yoursystem* is the name by which the remote system knows your system. This file is then executed by the remote system's copy of the command **uuxqt**.

command-string consists of a command name followed by zero or more arguments. Both the command's name and the arguments may be prefixed by a system name (sitename) and an exclamation mark. Note that all special characters must be escaped or enclosed in quotation marks to avoid being processed by your system's shell.

For example, the simplest form of the **uux** command is:

```
uux host!command arg0 ... argN
```

where *host* is the name of the remote system being contacted, as defined in file **/usr/lib/uucp/sys**, *command* is the name of the command to execute on the remote system, and *arg0* through *argN* are the arguments to *command*.

If an argument names a file, that file can reside on the remote system, on your system, or on some third system. For example, the command

```
uux widget!lp /usr/sally/herfile
```

asks site **widget** to print its own file **/usr/sally/herfile**. On the other hand, the command

```
uux widget!lp !$HOME/myfile
```

requests that site **widget** print on its line printer the file **myfile** from your home directory on your home system. Note that the '!' that prefixes **myfile** is shorthand for the name of your system. Finally, the command

```
uux widget!lp lepanto!/usr/fred/hisfile
```

requests that system **widget** print file **/usr/fred/hisfile**, which resides on the third site **lepanto**. If **widget** does not know how to contact site **lepanto**, the command fails.

If you wish, you can embed the shell operators '<', '>', ';', or '|' within a **uux** command. This lets you construct a more powerful command than you could do otherwise. Commands that contain these operators must be quoted, to ensure that your shell does not interpret them. For example, the command

```
uux "widget!pr /usr/sally/herfile > lepanto!~/fred/hisfile"
```

tells **uux** to use **pr** to format its file **/usr/sally/herfile**, and write the output into file **/usr/spool/uucppublic/fred/hisfile** on site **lepanto**. (Note that the tilde '~', as always, is a synonym for the home directory of the user that is executing the command; and a **uux** command is always executed by user **uucp** whose home directory is always **/usr/spool/uucppublic**.) Again, the command fails if you do not have appropriate permissions on **widget** or if **widget** does not have appropriate permissions on **lepanto**.

The operator '-' lets you use the standard input when constructing a **uux** command. For example, the command

```
who | uux - widget!lp
```

executes the **who** command on your system, pipes the output to **uux**, and tells **uux** to invoke the command **lp** on remote system **widget** to print the list of users on your system.

uux attempts to transfer any needed input files to the system that will be executing the requested command. You must enclose in parentheses any output files generated by *command*, to distinguish them from the names of input file.

Command-line Options

uux recognizes the following options:

-a *address*
Report the status of the job to *address*.

-C Copy local files to the spool directory.

-c Do not copy local files to the spool directory. This is the default. If the files are removed from their local directory before **uucico** processes them, the copy fails. The files must be readable by the **uucico** as well as the by the user who invokes **uux**.

-g *grade*
Set the grade of the file-transfer command. *grade* is a single ASCII character, from '0' to 'z'; the lower the ASCII value of *grade*, the more important the files.

-I *file* Read configuration information from *file* instead of from the default file **/usr/lib/uucp/sys**.

-j Print job identifiers on the standard output. **uux** creates a job identifier for each file-copying operation required to perform the operation. To cancel the copying of a file, pass the job identifier to the **uustat** with its option **-k**.

-l Link local files into the spool directory. If a file cannot be linked because it is on a different device, it is copied unless the **-c** option also appears (in other words, use of **-l** switches the default from **-c** to **-C**). If the files are changed before **uucico** processes them, the changed versions will be used. The files must be readable by the **uucico** as well as by the user who invoked **uux**.

-n Do not send mail about the status of a job, even if it fails. The default is to send mail to the requester should the command fail.

-

-p Read the standard-input device and pipe what is read into the command to be executed.

-r Queue the **uux** request but do not invoke **uucico** to perform the transfer. The default is to initiate **uucico**.

-x *event*
Log each *event* in the execution of **uux**, where *event* is one of the following values: **abnormal**, **config**, **spooldir**, or **execute**. A **-x** option can hold multiple events, each separated by commas; and a **uux** command line can hold more than **-x** option.

-z Notify requester should *command-string* fail.

Examples

The following script prints files on a remote system. The files named on the command line are sent unprocessed to system **prnsrvr** to be printed through that system's version of command **lp**. Option **-r** tells **uux** not to invoke **uucico** immediately, but merely spool the request for execution later.

```
for i in $*
do
      uux -r prnsrvr!lp !$i
done
```

Please note that the '!' that prefixes string "!$i" indicates that the file to be printed resides in the current directory on your home system.

The next example copies file **/foo** from system **george** and file **/bar** from system **norm** to your system and then invokes command **cmp** to compare their contents. It writes the results of the comparison into file **/tmp/cmp.results** on your local system:

```
uux -z "!cmp -l george!/foo norm!/bar >/tmp/cmp.results"
```

This command assumes that your system can talk to both **george** and **norm**, and that your system has permission to read file **/foo** on system **george** and file **/bar** on system **norm**. Option **-z** tells **uux** to send you mail when it has successfully completed the job.

The last example compiles file **mycode.c** on system **cserver**. The command redirects all of the compiler's error messages into file **/tmp/errors** on your local system:

```
uux 'cserver!cc -O -o (!mycode) !mycode.c > !/tmp/errors'
```

Note that the name of the output file **!mycode** is enclosed within parentheses. This is to protect the '!' from being interpreted by **uux**; it will be interpreted by **uuxqt** on the remote system.

See Also

commands, UUCP, uuxqt

Notes

You cannot pipe the output from a command on one system into a command on another. If *command-line* consists of several commands that are connected by pipe characters '|', only the first can be prefixed by a system name and '!'; every other command within the pipeline will be executed on the system named by the first command. For example, consider the command:

```
uux "mwc!wrap -w80 -t4 < !myfile.c | prps | lp"
```

This command passes file **myfile.c** from the current directory on your current system to command **wrap** on system **mwc** for processing; then pipes the output of **wrap** into **prps** on system **mwc** for transformation into PostScript; and then pipes the output of **prps** into **lp**, again on system **mwc**, for printing. If you embed a '!' within the subsequent commands of a pipeline, **uux** will expand it into something quite unexpected (and probably unwelcome).

It is not a good idea to use the metacharacter '*' within *command-line*. The odds that it will be expanded into what you want are very small.

Every command that you spool with **uux** is executed within a special execution directory on the remote system. (Under COHERENT, this directory is **/usr/spool/uucp/.Xqtdir**; it may vary on other systems.) Before it executes the command, UUCP copies into that special directory each file that the command names, unless that file already resides on the system within which the command is being executed. For this reason, each file named in a **uux** command must be unique, regardless of its full path name. For example, the following command will not work:

```
uux  "marian!diff fred!/x/testfile ivan!/y/testfile > !xyz.diff"
```

It fails because **uux** (or, to be more accurate, **uuxqt**) copies file **testfile** from system **fred** into its execution directory, then copy **testfile** from system **ivan** into the test directory. The second copied **testfile** overwrites the first, and thus the command **diff** fails.

uux was written by Ian Lance Taylor (ian@airs.com).

uuxqt — Command

Execute commands requested by a remote system
uuxqt

uuxqt reads files from directory **/usr/spool/uucp/***sitename*, and executes them. It recognizes the files to execute (as opposed to the files that simply contain data, such as mail messages) because they are prefixed with the string "X.". **uuxqt** executes only the programs for which the remote system has permission.

uuxqt is invoked by either **uucp** or **uucico**. It is not generally considered a user-callable program.

Command-line Options

uuxqt recognizes the following command-line options:

-c *command*
Only execute *command*; ignore requests to execute any other command. For example:

```
uuxqt -c rmail
```

-s *system*
Only execute requests originating from *system*.

-I *file* Read configuration information from *file* instead of from **/usr/lib/uucp/sys**.

-x *activity*
Log each *activity*; *activity* must be one following: **abnormal**, **config**, **spooldir**, and **execute**. An **-x** option can name more than one activity, with activities being separated by commas; and a **uuxqt** command-line can have more than one **-x** option.

Files

/usr/lib/uucp/config — Configuration file
/usr/spool/uucp/*sitename* — Directory for execute files
/usr/spool/uucp/Log — UUCP log file
/usr/spool/uucp/Debug — Debugging file

See Also

commands, uucico, UUCP, uucp, uux

Notes

uuxqt was written by Ian Lance Taylor (ian@airs.com).