NAME

enable disable dialin dialout – control login lines

SYNOPSIS

/etc/enable [**-deio**] *terminal*
/etc/disable [**-deio**] *terminal*
/etc/dialin [**-deio**] *terminal*
/etc/dialout [**-deiorw**] *terminal*

DESCRIPTION

The **enable** family of programs is used to change the contents of */etc/ttys* and tell **init**(8) about it in a controlled way. The operation can be governed both by switches and by the program name.

disable (or -d)	Disables a terminal for login. Can only be used by the superuser.
enable (or -e)	Enables a terminal for login. Can only be used by the superuser.
dialout (or -o)	Disables a terminal for login. Can be used by anybody if the terminal is marked as public in */etc/ttys*. The ownership of the device is changed to that of the user issuing the dialout command. The protection on the device are by default set to -rw-------, but the **-r** and **-w** switches can change this. These switches determine if other users should be allowed to read respectively write to the device. An entry (users name in upper case) is also made in the */etc/utmp* file so that the **who**(1) command shows that the line is in use.
dialin (or -i)	Enables a terminal for login. Can only be used by the user who allocated the line with the **dialout** command, or by the superuser.

NOTES

If a terminal both has a */dev/tty*XX and a */dev/cua*XX device then that terminal is assumed to be a modem and that the *tty* device has a minor number that is 128 higher than usual. The *cua* device is assumed to have the standard minor number. **enable** will use the *cua* device when it must open the device without carrier present.

This version of **enable** is **EXPERIMENTAL.**

EXAMPLES

```
dialout tty02
who
ls -l /dev/tty02
kermit -l /dev/cua02 -b 2400 -p
dialin tty02
```

FILES

/etc/ttys	terminals and their flags
/etc/utmp	notes to the world
/dev/*	terminals
/dev/cua*	dialers

BUGS

If the */etc/ttys* file is changed while this program is running, unpredictable things may happen.

SEE ALSO

login(1), **getty**(8), **ttys**(5), **who**(1), **init**(8)

DIAGNOSTICS

Several diagnostics can be issued. You should (as usual :-)) have no problems to understand them.

NAME

login – log in to the system

SYNOPSIS

login [**–p**] [*username*]

DESCRIPTION

login signs *username* on to the system initially. **login** may also be used at any time to change from one userID to another.

When used with no argument, **login** requests a user name and password (if appropriate). Echoing is turned off (if possible) while typing the password. Note: the number of significant characters in a password is 8. (See **passwd**(1).)

When successful, **login** updates accounting files, prints the message of the day, informs you of the existence of any mail, and displays the time you last logged in. If failed login attempts have been made since your last login, a message about this will be printed. None of these messages are printed if there is a *.hushlogin* file in your home directory; this is mostly used to make life easier for nonhuman users, such as **uucp**(1).

login initializes the user and group IDs and the working directory, then starts a command interpreter shell (usually either **/bin/sh**, **/bin/ksh** or **/bin/csh**) according to specifications found in the file */etc/passwd*. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash ('–') prepended.

If the command interpreter is specified as ''*'', then **login** uses the home directory specification as a new root. It changes to that directory and issues the **chroot**(2) system call. Note that all file specifications from now on refers to the new root. This new root should have an */etc* directory with a new *passwd* file. **login** uses this new *passwd* file to locate the users real home directory and command interpreter. The new root should also contain suitable programs in the new */bin* directory, especially the users command interpreter and application. This feature can be used for security reasons to isolate certain users from the rest of the system, for example accounts without passwords that run specific applications.

login also modifies the environment (**environ**(5)) with information specifying home directory, command interpreter, your username, default search path and mailbox. The **–p** argument preserves the remainder of the environment, otherwise any previous environment is discarded.

If the file */etc/nologin* exists, **login** prints its contents on the user's terminal and exits. This is used by **shutdown**(8) to stop logins when the system is about to go down.

The **login** command, recognized by **sh**(1), **ksh**(1) and **csh**(1), is executed directly (without forking), and terminates that shell. To resume working, you must log in again.

login times out and exits if its prompt for input is not answered within a reasonable time.

When the Bourne shell **(sh)** and the Korn shell **(ksh)** starts up, it reads a file called *.profile* from your home directory (that of the username you use to log in). When the C shell **(csh)** starts up, it reads a file called *.cshrc* from your home directory, and then reads a file called *.login*.

NOTE

If the */usr/adm/lastlog* file does not exist, no information about the last login will be printed. This file can be created by the superuser with the command ''touch /usr/adm/lastlog''.

OPTIONS

−p Preserve any existing environment variables and their values; otherwise the previous environment is discarded.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/adm/lastlog	time of last login and login failure counters
/etc/gettydef	terminal parameters and prompts
/usr/spool/mail/*	the post office
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stop login, print message
/bin/sh	antique command interpreter
/bin/ksh	excellent command interpreter
/bin/csh	braindamaged command interpreter
~/.profile	startup file for **sh** and **ksh**
~/.cshrc	initialization file for **csh**
~/.login	startup file for **csh**
~/.hushlogin	makes login quieter

SEE ALSO

sh(1), **ksh**(1), **csh**(1), **mail**(1), **passwd**(1), **uucp**(1), **passwd**(5), **environ**(5), **gettydef**(5), **utmp**(5), **init**(8), **getty**(8), **shutdown**(8), **mkgettydef**(8), **lastlogins**(8)

DIAGNOSTICS

No directory Login denied
Your home directory does not exist, contact the system administrator.

Login incorrect
If the name or the password is bad (or mistyped).

number **failures since last login**
Contact your system administrator if you did not cause this.

No shell

The command interpreter in the passwd file could not be started. Contact your system administrator.

Timeout period expired

You are to slow.

NAME

/etc/ttys – login terminals file

DESCRIPTION

The */etc/ttys* file contains a list of the devices that are candidates for logins. **init**(8) uses this file at startup to start login processes.

The file contains entries of the form

state public name

A name must be the filename of a device special file. The path is assumed to be */dev/* so that string should not be supplied. If *state* is ''1'', the device is enabled for logins; if ''0'', the device is disabled. If *public* is ''P'', the device is public and any user may allocate it using the **dialout** command (see **enable**(8).) If *public* is ''0'', the device belongs to the system.

EXAMPLES

The entry ''10tty01'' means that a login process should be started on tty01. The entry ''0Ptty02'' means that the tty02 device should be free for anybody to use.

FILES

/etc/ttys

SEE ALSO

enable(8), **login**(1), **getty**(8), **ttys**(5), **init**(8)

NAME
enable disable dialin dialout – control login lines

SYNOPSIS
/etc/enable [**-deio**] *terminal*
/etc/disable [**-deio**] *terminal*
/etc/dialin [**-deio**] *terminal*
/etc/dialout [**-deiorw**] *terminal*

DESCRIPTION
The **enable** family of programs is used to change the contents of */etc/ttys* and tell **init**(8) about it in a controlled way. The operation can be governed both by switches and by the program name.

disable (or -d)	Disables a terminal for login. Can only be used by the superuser.
enable (or -e)	Enables a terminal for login. Can only be used by the superuser.
dialout (or -o)	Disables a terminal for login. Can be used by anybody if the terminal is marked as public in */etc/ttys*. The ownership of the device is changed to that of the user issuing the dialout command. The protection on the device are by default set to -rw-------, but the **-r** and **-w** switches can change this. These switches determine if other users should be allowed to read respectively write to the device. An entry (users name in upper case) is also made in the */etc/utmp* file so that the **who**(1) command shows that the line is in use.
dialin (or -i)	Enables a terminal for login. Can only be used by the user who allocated the line with the **dialout** command, or by the superuser.

EXAMPLES
```
dialout tty02
who
ls -l /dev/tty02
kermit -l /dev/tty02 -b 2400 -p
dialin tty02
```

FILES
/etc/ttys	terminals and their flags
/etc/utmp	notes to the world
/dev/*	terminals

BUGS
If the */etc/ttys* file is changed while this program is running, unpredictable things may happen.

SEE ALSO
login(1), **getty**(8), **ttys**(5), **who**(1), **init**(8)

DIAGNOSTICS
Several diagnostics can be issued. You should (as usual :-)) have no

problems to understand them.

NAME

getty – adjust terminal line and start login

SYNOPSIS

getty *tty*

DESCRIPTION

getty waits for input on the *tty* device. When correct terminal parameters have been determined, **getty** asks for a username and starts **login**(1). **getty** can be told to select between a set of speeds or to automatically determine the speed from the users input. The behaviour of **getty** can be changed with the **mkgettydef**(8) program. From the list of customizable items some can be noted: heading, prompt and terminal parameters.

FILES

/etc/gettydef	terminal parameters and prompts
/bin/login	completes the login process

SEE ALSO

login(1), **init**(8), **mkgettydef**(8)

DIAGNOSTICS

Timeout period expired

You are to slow.

NAME

init - process control initializer

SYNOPSIS

/etc/init

DESCRIPTION

init is started by the kernel directly after boot. After start, init checks the autoboot flag, and if it is set to ''NO'', starts the default command interpreter (*/bin/sh*) with the console as the controlling terminal. This mode is called **single user mode**. When the system administrator exits this command interpreter, with ^D, init begins to enter **multi user mode**. If the autoswitch is set to ''YES'' then multi user mode is entered without going through single user mode. The actions needed to bring up multi user mode starts with the interpretation of the Run Command file (*/etc/rc*) by the standard command interpreter. After that, init looks through the */etc/ttys* file and forks of an login process for each terminal that is enabled. This is done by executing */etc/getty* with the terminal name as the first argument. Now init starts to sleep. Each time a process associated with a terminal dies, init starts a new login process.

init will also wake up if certain signals are received.

SIGINT	can be used to tell init that a command has been placed in shared memory, the default beeing to force a re-examination of the */etc/ttys* file (in case it has been changed)
SIGHUP	can be used to return to single user mode
SIGQUIT	is used to suppress creations of login processes as users loggs out
SIGTERM	halts the system

FILES

/etc/ttys	terminals and their flags
/bin/sh	standard command interpreter used for single user shell and Run Command file interpretation
/etc/getty	waits for terminal activity, then executes /bin/login
/etc/rc	Run Command file
/etc/utmp	accounting
/usr/adm/wtmp	accounting
/etc/mtab	mounted file systems
/dev/console	controlling terminal for single user mode
/dev/autosw	flags autoboot or single user mode
/dev/*	terminals
/usr/adm/messages	diagnostics

BUGS

The signals used for various actions has different functions on all other UNIX systems. This will probably change.

SEE ALSO

login(1), **getty**(8), **stty**(1), **ttys**(5), **kill**(1), **kill**(2), **shutdown**(8), **enable**(8)

DIAGNOSTICS

Diagnostics are written to the system messages file by the *log* process.

INIT RECEIVED UNSOLICITED SIGNAL *number*
some application is sending bogous signals to init

Init failed to execute '*command*', sleeping
init could not start a login process

NAME

lastlogins – display times and failures for logins

SYNOPSIS

/etc/lastlogins

DESCRIPTION

lastlogins displays a list of usernames together with times and terminals for the last login session. Also the failure counters are displayed. The first failure figure is the total failures since the */usr/adm/lastlog* file was created, the second figure is the failures since that user logged in last time.

FILES

/usr/adm/lastlog	information source

SEE ALSO

login(1)

DIAGNOSTICS

can not open /usr/adm/lastlog

NAME

mkgettydef – create definition file for getty and login

SYNOPSIS

/etc/mkgettydef *gettydef.src*

DESCRIPTION

mkgettydef is the compiler that compiles the file named as it's argument into a definition file for getty and login. This file contains terminal characteristics, prompt texts and baud rate detection strategy.

The structure of the source input contains one block for each device used for login.

device *name*
definitions for device name

Device names are selected from **default**, **network** or a device name listed in */etc/ttys*. The default device is a pseudo device that supplies data not given for other devices. The network device is used for logins over the DNET network.

The device definitions are built from the following keywords.

herald	Takes a string as argument. This string is displayed before **getty**(8) issues the first login prompt. Default value is an empty string.
loginprompt	Takes a string as argument. This string is the login prompt issued by **login**(1). Default value is "login: "
gettyprompt1	Takes a string as argument. This string is the login prompt issued by **getty**. Default value is "login: "
gettyprompt2	Takes a string as argument. This string is the login prompt issued by **login** when called from **getty**. Default value is "login: "
passwdprompt	Takes a string as argument. This string is the password prompt issued by **login**. Default value is "password:"
timeout	Takes an integer as argument. This integer is the timeout time in seconds before **getty** or **login** terminates. Default value is 60 seconds.
stty	Takes a string as argument. The string contains **stty**(1) commands describing the terminal characteristics. Default is all stty flags off.
strategy	This keyword determines how the speed should be selected. The argument is one of the keywords **normal**, **autobaud** or **rotate**. Normal strategy takes speed from the stty entry. Autobaud determines the baud from the input. The user should enter carriage returns until the prompt shows. The algorithm used by **getty** can select between 300, 600, 1200, 2400,

4800 and 9600 baud. Rotate strategy takes up to four more arguments. These arguments are speeds that are selected in order each time the user hits BREAK.

trigger This keyword is unique to the network device. The argument should be a string that describes the beginning of the network pseudo terminal. If, at login time, **getty** finds that the input comes from a trigger terminal then no terminal parameters should be touched as they are sent over from the calling machine. The **stty** keyword is illegal for the network device.

A source line beginning with ''#'' is taken as a comment. The normal escape notation used in ''C'' can be used. See example.

EXAMPLE

```
#
# definitions for getty and login
#

#
# default values for all devices, may be overridden selectively below
#

device default

herald       "\r\n    ABCenix 5.18  (Hubert)\r\n\r\n"
loginprompt  "Login: "
gettyprompt1 "Hubert login: "
gettyprompt2 "Hubert login: "
passwdprompt "Password: "
timeout      60
stty    "-ignbrk brkint ignpar -parmrk -inpck istrip -inlcr     \
        -igncr icrnl -iuclc ixon ixany -ixoff                  \
        opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel \
        nl0 cr0 tab3 bs0 vt0 ff0                               \
        9600 cs7 -cstopb cread parenb -parodd hupcl clocal     \
        isig icanon -xcase echo echoe echok -echonl -noflsh    \
        intr '^?' quit '^\\' erase '^H' kill '^X' eof '^D' eol '^-'"
strategy normal

#
# unique definitions for device network
#

device network
trigger "/dev/pk"

herald "\r\n    ABCenix 5.18  (Hubert) (network)\r\n\r\n"
strategy normal
```

```
#
# unique definitions for device console (workstation screen)
#

device console

herald "\r\n    ABCenix 5.18  (Hubert) (console)\r\n\r\n"
strategy normal

#
# unique definitions for device tty02 (dialin/dialout modem)
#

device tty02

herald "\r\n    ABCenix 5.18  (Hubert) (tty02)\r\n\r\n"
stty "-clocal"
strategy autobaud

#
# unique definitions for device tty03 (local terminal)
#

device tty03

herald "\r\n    ABCenix 5.18  (Hubert) (tty03)\r\n\r\n"
strategy rotate 9600 4800 2400
```

FILES

gettydef	result
gettydef.src	standard source text
/etc/ttys	terminals

SEE ALSO

login(1), **getty**(8), **stty**(1), **ttys**(5)

DIAGNOSTICS

Error checking is excessive and the messages are intended to be self explanatory (as they say :-))

HNIX Init/Login replacement

Linköping 890313

Jag hoppas att du blir nöjd med detta programpaket.
Dessa program utvecklades ursprungligen för mitt eget bruk då jag fann att de medlevererade programmen inte uppfyllde de enkla krav som jag ställde på dem. Några av dessa krav var automatisk hastighetsanpassning vid inloggning, en init som inte krashar systemet när man startar och stoppar login med hjälp av enable och disable, en login som inte självsvänger och lastar ner terminal-drivaren när ett modem sätter DCD hög, mm.
När andra personer fick reda på vad jag gjort och uttryckte önskemål om en "distribution" så framställde jag dokumentation och snyggade upp lite bitar.
Jag förutsätter att du anser att värdet på programmen överstiger det pris jag begär för dem och att du därför inte uppmuntrar att kopior görs av programmen.
Jag kan naturligtvis inte förhindra att kopior görs, men om jag finner att många kopior blir gjorda tolkar jag detta som att programmen inte är värda det pris jag begär och därmed minskar chansen att jag producerar fler programpaket liknande detta.

Nog om detta, lycka till med installationen.

Göran

Hnix init / login replacement

Document: /usr/src/man/hnix.mm

Author: Goran Larsson, HoH

Date: March 13, 1989

3:rd Edition

Abstract

This document describes the configuration and installation of the programs and files that together form the *Hnix init/login replacement*.

CONTENTS

3:rd Edition March 13, 1989

1. **Overview**

□ This package contains replacaments for the following **abcenix** programs.

- init
- login
- enable
- disable

The replacements provides more functionality compared with the old programs. **getty** and **login** for example replaces the old **login**, and has customizable prompts and supports automatic baudrate detection. **enable/disable** when named **dialin/dialout** can be used by unprivileged users on selected terminal lines. **init** is more stable and does not die when it receives unwanted signals. **init** also uses an alternate method for communications with **enable/disable/dialin/dialout**. This method passes commands through shared memory and provides feedback so that the calling program can detect when **init** has completed the command. The **key** used by this mechanism is *0x49*, so this **key** can not be used for other purposes.

□ Manual pages are provided for all programs.

□ The package also provides additional programs.

2. Saving old files

□ Backup the following files.

- /etc/init
- /bin/login
- /bin/enable
- /bin/disable
- /etc/ttys

You can copy them into a save directory or copy (tar) them to floppy.

3. **Configuration**

□ Read the package into a scratch directory.

- cd /usr
- tar xvf /dev/mf2
- cd /usr/hnix

You should have extracted the following files:

□ Install
□ Headers
□ enable
□ enable-HUP
□ freebee/
□ getty
□ gettydef.src
□ init
□ initlib/
□ lastlogins
□ login
□ mkgettydef
□ shutdown
□ sysname
□ ttys

□ Edit the *ttys* file to match your system. Refer to the **ttys**(5) manual page.

□ Read the **mkgettydef**(8) manual page and modify *gettydef.src* to taste. You can leave out the network device if you do not run **abcnet** or **dnet**, although it does not hurt to keep it.

□ Read the boxed comment on the front page regarding guaranties and backing up your disks.

4. **Installation**

□ Become superuser and complete the installation by executing the install script named **Install**. Due to some unknown reason root has been given the group 50 in the */etc/passwd* file. This is not good since many programs assume that root belongs to group zero. If you look in */etc/group* you will find that group zero is called root. Just change the fragment ":0:50:" to ":0:0:" in */etc/passwd*. If this is not done, then **enable/disable** will not work. You may want to change the **Install** script if your executable files are owned by somebody other than **bin**, for example **sys**. There might be other things you might like to change, but keep security in mind when changing permissions on the programs and datafiles.

- ./Install

□ Reboot and enjoy.

5. **Ownerships and protections**

The following table can be used to check that the files are owned by the correct owner and have the right protection.

file	owner	group	protection	note
/etc/init	*bin*	*bin*	*710*	
/etc/getty	*bin*	*bin*	*710*	
/bin/login	*root*	*root*	*4711*	*set uid*
/etc/mkgettydef	*bin*	*bin*	*710*	
/etc/enable	*root*	*root*	*6711*	*set uid & gid*
/etc/disable	*root*	*root*	*6711*	*set uid & gid*
/etc/dialin	*root*	*root*	*6711*	*set uid & gid*
/etc/dialout	*root*	*root*	*6711*	*set uid & gid*
/etc/lastlogins	*bin*	*bin*	*710*	
/etc/sysname	*bin*	*bin*	*710*	
/etc/shutdown	*bin*	*bin*	*710*	
/etc/ttys	*root*	*root*	*664*	
/etc/gettydef.src	*root*	*root*	*660*	
/etc/gettydef	*root*	*root*	*664*	
/usr/adm/lastlog	*root*	*root*	*660*	

6. **Experimantal SIGHUP Support**

This version of the *Hnix init/login replacement kit* has an experimental version of **enable/disable/dialin/dialout** that should be used if you want the terminal driver to send **SIGHUPs** to login processes that users leave running by hanging up the phone without logging out. The steps needed to use this **EXPERIMENTAL** feature is

- ☐ Rename the modem device from *tty02* to *cua02*.
- ☐ Create a new modem device with the command **/etc/mknod /dev/tty02 c 1 130**.
- ☐ Save the ''old'' enable and then copy enable-HUP to /etc/enable. Check the links, owner, and protections.
- ☐ Make sure that the */etc/gettydef.src* file specifies *stty* "*-clocal*" for the **tty02** device. If not, edit and recompile.

If you want to dial out using, say, **kermit** you should use the *cua* device, otherwise **kermit** will be unable to open the device. The proper sequence is

- /etc/dialout tty02
- kermit -l /dev/cua02 -b 2400 -p e
- /etc/dialin tty02

Once the modem has set **DCD** to true, you can use the *tty* device if you like. Note that the modem must drop **DCD** when it looses the carrier, otherwise all this will simply not work. The cable connecting the modem to the port must have the proper wires., it should be a straight cable, male in one end, female in the other, and pins 2 to 8 and 20 connected and pin 1 connected to the screen.

7. **Versions**

The files that together makes this edition has individual version numbers. If you execute the command **strings** *program* **| grep Header:** then a string like

$Header: init.c,v 1.13 89/03/13 00:55:53 root Exp $

appears. A complete list of these version numbers can be found in the *Headers* file. No guaranties are given that other versions of these programs may work together.

8. Initlib

The directory *initlib* contains sources for the interface to **init**. If you want to write your own program that does things like **shutdown** or **enable**, this is for you. Note that missuse of init can make your system behave very strange. Be sure that you know what you are doing.

9. **FreeBees**

The directory *freebee* contains some programs that are totally unrelated to this software package. A short description is all that you get, except for the sources.

stat	Sort of a *doit yourself ls*. It can display file information in any way you want. A man-page is included.
graphmem	A demonstration program that show how to access the graphic memory from C.
wipe	This program clears the screen graphically (all pixels)
cache	A variation of **graphmem** that dumps the font cache on the screen.
wipec	This program clears the font cache. All fonts are invisible after this!
unloadfont	This program unloads a font from the fontcache. This is needed if you have linked in a new font in the *used* directory and the old font on that position is already loaded. Run this program and then load the new font with the appropriate escape sequence.
noclick	This little goodie will keep you sane. It kills the horrible keyboard click Luxor want's us to hear and hate.

10. **Problems**

The only known problem at this time is that a warning message

Log info out of phase, info may be lost...

may be emitted by the **log** daemon when **shutdown** is used to take the system down to single user mode and the system is then rebooted from single user mode using **^D**. Log info is not lost, so for the time beeing, just ignore the message.

NAME

shutdown – allow superuser to bring the system down gracefully

SYNOPSIS

/etc/shutdown [**-krhfn**] *shutdowntime* [*message*]

DESCRIPTION

shutdown allows super users to tell users and remind users of iminent shutdown of the unix system and shut it down automatically and even reboot or halt the machine if they desire. The *shutdowntime* can be given as either an absolute time in the hour:minute format, or a relative time in the +minutes format. Immediate shutdown can be specified by the time specification ''now''. After nagging all users for a while, **shutdown** will disable logins at most five minutes before actual halt. When halt time arrives, all processes are killed (SIGHUP followed by SIGKILL) and all filesystems are dismounted.

OPTIONS

-k	fake shutdown, make users think the system is going down
-r	reboot
-h	halt
-f	fast boot
-n	no sync before going down

Enters single user mode without -r or -h options.

EXAMPLES

/etc/shutdown 17:00 preventive maintenance
/etc/shutdown -h now
/etc/shutdown -k 10:00 backup of all disks

FILES

/etc/nologin	created to cause **login**(1) to disable logins
/fastboot	created if -f option is given and can be used by **/etc/rc**

SEE ALSO

login(1), **init**(8)

DIAGNOSTICS

Some messages may be given when problems shows up.

NOTE

As it is impossible to halt an unmodified ABC 1600 due to the watchdog, the halt and reboot options both reboots. If the watchdog has been disabled, both options halt the system. It is not sure whether the nosync option works, it might be so that the filesystem handlers always syncs before exiting. The fast boot option is of little use in abcenix as fsck is invoked automatically, if needed. To simplify it: it is only the fake shutdown (-k) and reboot (-r) options that are of interrest.

BUGS

The warning message *Log info out of phase, info may be lost...* may be emitted by the **log** daemon when **shutdown** is used to take the system down to single user mode and the system is then rebooted from single user mode using **^D**.

NAME

sysname – change systems node name

SYNOPSIS

/etc/sysname *name*

DESCRIPTION

sysname patches the kernel so that the **uname**(1) command returns the correct name. **sysname** should be run from the */etc/rc* file.

EXAMPLES

```
uname -a
/etc/sysname hubert
uname -a
```

FILES

/abcenix	file used as namelist
/etc/kmem	file used as core file (patched)

SEE ALSO

uname(1), **uname**(2)

DIAGNOSTICS

```
error opening /etc/kmem
error in /abcenix namelist
symbol not in /abcenix
error reading /etc/kmem
error writing /etc/kmem
```

NAME
sh, rsh - shell, the standard/restricted command programming language

SYNOPSIS
sh [**−acefhikmnorstuvx**] [**−o** option] ... [arg ...]
rsh [**−acefhikmnorstuvx**] [**−o** option] ... [arg ...]

DESCRIPTION
Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Definitions.
A *metacharacter* is one of the following characters:

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

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

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

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

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent Shell using the **-p** option of the special commands **read** and **print** described later. Only one such command can be active at any given time. The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command.

These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the identifier is read and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The **-h** option of the **set** command makes each command name which is an *identifier* into a tracked alias.

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

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

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

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

Command Substitution.
The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed. The command substitution `**cat file**` can be replaced by the equivalent but faster `**<file**`. Command'substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

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

shells but only values to the environment.

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

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

> *name*=*value* [*name*=*value*] . . .

If the integer attribute, **−i**, is set for *name* the *value* is subject to arithmetic evaluation as described below.
Positional parameters, parameters denoted by a number, may be assigned values with the **set** special command. Parameter **$0** is set from argument zero when the shell is invoked.
The character **$** is used to introduce substitutable *parameters*.

${*parameter***}**
: The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is a digit then it is a positional parameter. If *parameter* is ***** or **@**, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). If an array *identifier* with subscript ***** or **@** is used, then the value for each of the elements is substituted (separated by spaces).

${#*parameter***}**
: If *parameter* is not *****, the length of the value of the *parameter* is substituted. Otherwise, the number of positional parameters is substituted.

${#*identifier***[*]}**
: The number of elements in the array *identifier* is substituted.

${*parameter***:−***word***}**
: If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**
: If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
: If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter***:+***word***}**
: If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

${*parameter***#***pattern***}**
${*parameter***##***pattern***}**

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

${*parameter***%***pattern***}**
${*parameter***%%***pattern***}**

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

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

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

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

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

– Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last executed command.

$ The process number of this shell.

_ The last argument of the previous command. This parameter is not set for commands which are asynchronous.

! The process number of the last background command invoked.

PPID The process number of the parent of the shell.

PWD The present working directory set by the **cd** command.

OLDPWD
The previous working directory set by the **cd** command.

RANDOM
Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.

REPLY
This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.

The following parameters are used by the shell:

CDPATH
The search path for the *cd* command.

COLUMNS
If this variable is set, the value is used to define the width

of the edit window for the shell edit modes and for printing **select** lists.

EDITOR
If the value of this variable ends in *emacs*, *gmacs*, or *vi* and the **VISUAL** variable is not set, then the corresponding option (see Special Command **set** below) will be turned on.

ENV If this parameter is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the *shell* is invoked. (See *Invocation* below.) This file is typically used for *alias* and *function* definitions.

FCEDIT
The default editor name for the **fc** command.

IFS Internal field separators, normally **space**, **tab**, and **new-line** that is used to separate command words which result from command or parameter substitution and for separating words with the special command **read**.

HISTFILE
If this parameter is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See *Command re-entry* below.)

HISTSIZE
If this parameter is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

HOME The default argument (home directory) for the **cd** command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, then the shell informs the user of arrival of mail in the specified file.

MAILCHECK
This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds. If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set then the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a **?** and a message that will be printed. The message will undergo parameter and command substitution with the parameter, **$_** defined as the name of the file that has changed. The default message is *you have mail in* $_.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh* (except in *.profile*).

PS1 The value of this parameter is expanded for paramter substitution to define the primary prompt string which by default is ''**$** ''. The character **!** in the primary prompt string is replaced by the *command* number (see *Command Re-entry* below).

PS2 Secondary prompt string, by default ''> ''.

PS3 Selection prompt string used within a **select** loop, by default ''**#?** ''.

SHELL
The pathname of the *shell* is kept in the environment. At invocation, if the value of this variable contains an **r** in the basename, then the shell becomes restricted.

TMOUT
If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)

VISUAL
If the value of this variable ends in *emacs*, *gmacs*, or *vi* then the corresponding option (see Special Command **set** below) will be turned on.

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

Blank Interpretation.

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

File Name Generation.

Following substitution, each command *word* is scanned for the characters *****, **?**, and **[** unless the **−f** option has been **set**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character **.** at the start of a file name or immediately following a **/**, as well as the character **/** itself, must be matched explicitly. In other instances of pattern matching the **/** and **.** are not treated specially.

***** Matches any string, including the null string.

? Matches any single character.

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

Quoting.

Each of the *metacharacters* listed above (See *Definitions* above). has a special meaning to the shell and cause termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\\new-line** is ignored. All characters enclosed between a pair of single quote marks (**' '**), except a single quote, are quoted. Inside double quote marks (**" "**), parameter and command substitution occurs and \ quotes the characters \, **'**, **"**, and **$**. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2"** ....

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

Arithmetic Evaluation.

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

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

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

Prompting.

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

Input/Output.

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[−]*word* The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\new-line** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If − is appended to <<, then all leading tabs are stripped from *word* and from the document.

<&*digit* The standard input is duplicated from file descriptor *digit* (see *dup*(2)). Similarly for the standard output using **>&** *digit*.

<&− The standard input is closed. Similarly for the standard output using **>&−**.

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

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

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

... 1>*fname* 2>&1

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

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

Environment.
The *environment* (see *environ*(7)) is a list of name-value pairs that is

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

The environment for any *simple-command* or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word of the form *identifier=value*. Thus:

```
     TERM=450 cmd args                    and
     (export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

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

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

Functions.

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

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

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

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

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

Jobs.

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

```
[1] 1234
```

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

This paragraph and the next require features that are not in all versions of UNIX and may not apply. If you are running a job and wish to do something else you may hit the key **^Z** (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A **^Z** takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

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

There are several ways to refer to jobs in the shell. The character **%** introduces a job name. If you wish to refer to job number 1, you can name it as **%1** . Jobs can also be named by prefixes of the string typed in to **kill** or restart them. Thus, on systems that support job control, '**fg %ed**' would normally restart a suspended *ed*(1) job, if there were a suspended job whose name began with the string 'ed'.

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

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

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals.
The INT and QUIT signals for an invoked command are ignored if the command is followed by **&** and job **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution.
Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a **return**, the positional parameter list is restored and any trap set on **EXIT** within the function is executed. The value of a *function* is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a *special command* or a user defined *function*, a process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between colon delimiters, or at the end of the path list. If the command name contains a **/** then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and named parameters are removed in this case. A parenthesized command is also executed in a sub-shell.

Command Re-entry.
The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.history** is used if the **HISTFILE** variable is not set or is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to **fc** then the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined then */bin/ed* is used.

The edited command(s) is printed and re-executed upon leaving the editor. The editor name – is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if **r** is aliased to **'fc -e -'** then typing '**r bad=good c**' will re-execute the most recent command which starts with the letter **c**, replacing the string **bad** with the string **good**.

In-line Editing Options

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

The editing features require that the user's terminal accept 'RETURN' as carriage return without line feed and that a space (' ' must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to 'space'. Hewlett-Packard series 2621 terminal users should set the straps to 'bcGHxZ etX'.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it is defined, otherwise 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, *****) if the line extends on the right (left, both) side(s) of the window.

Emacs Editing Mode

This mode is entered by enabling either the *emacs* or *gmacs* option. The only difference between these two modes is the way they handle **^T**. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, **^F** is the notation for control **F**. This is entered by depressing 'f' while holding down the 'CTRL' (control) key. The 'SHIFT' key is *not* depressed. (The notation **^?** indicates the DEL (delete) key.)

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

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

^F	Move cursor forward (right) one character.

M-f	Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the stty command, usually **^H** or **#**.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is **^?** (DEL, the default) then this command will not work).
^T	Transpose current character with next character in *emacs* mode. Transpose two previous characters in *gmacs* mode.
^C	Capitalize current character.
M-C	Capitalize current word.
^K	Kill from the cursor to the end of the line. If given a parameter of zero then kill from the start of line to the cursor.
^W	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User defined kill character as defined by the stty command, usually **^G** or **@**.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(Null character) Set mark.
M-	(Meta space) Set mark.
^J	(New line) Execute the current line.
^M	(Return) Execute the current line.
eof	End-of-file character, normally **^D**, will terminate the shell if the current line is null.
^P	Fetch previous command. Each time **^P** is entered the previous command back in time is accessed.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
^N	Fetch next command. Each time **^N** is entered the next command forward in time is accessed.
^R*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given the search is forward. *String* is terminated by a "RETURN" or "NEW LINE".
^O	Operate - Execute the current line and fetch the next line relative to current line from the history file.
M-*digits*	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a

	parameter are **^F**, **^B**, *erase*, **^D**, **^K**, **^R**, **^P** and **^N**.
M-*letter*	Soft-key - Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the line. The *letter* must not be one of the above meta-functions.
M-_	The last parameter of the previous command is inserted on the line.
M-.	The last parameter of the previous command is inserted on the line.
M-*	Attempt file name generation on the current word.
^U	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill and interrupt (normally **^?**) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
^V	Display version of the shell.

Vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC (**033**) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command.
When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can than modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.
If the option **viraw** is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

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

Motion Edit Commands

These commands will move the cursor.

[*count*]**l**	Cursor forward (right) one character.
[*count*]**w**	Cursor forward one alpha-numeric word.
[*count*]**W**	Cursor to the beginning of the next word that follows a blank.

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

Search Edit Commands

These commands access your command history.

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

Text Modification Edit Commands

These commands will modify the line.

a	Enter input mode and enter text after the current character.
A	Append text to the end of the line. Equivalent to **$a**.

[*count*]**c***motion*
c[*count*]*motion*

Delete current character through the character *motion* moves the cursor to and enter input mode. If *motion* is **c**, the entire line will be deleted and input mode entered.

C	Delete the current character through the end of line and enter input mode. Equivalent to **c$**.
S	Equivalent to **cc**.
D	Delete the current character through the end of line.

[*count*]**d***motion*

d[*count*]*motion*
Delete current character through the character *motion* moves the cursor to. Equivalent to **d$**. If *motion* is **d** , the entire line will be deleted.

i	Enter input mode and insert text before the current character.
I	Insert text before the beginning of the line. Equivalent to the two character sequence **^i**.
[*count*]**P**	Place the previous text modification before the cursor.
[*count*]**p**	Place the previous text modification after the cursor.
R	Enter input mode and replace characters on the screen with characters you type overlay fashion.
r*c*	Replace the current character with *c*.
[*count*]**x**	Delete current character.
[*count*]**X**	Delete preceding character.
[*count*]**.**	Repeat the previous text modification command.
~	Invert the case of the current character and advance the cursor.
[*count*]**_**	Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.
*****	Causes an ***** to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

Other Edit Commands
Miscellaneous commands.

u	Undo the last text modifying command.
U	Undo all the text modifying commands performed on the line.
[*count*]**v**	Returns the command **fc -e ${VISUAL:-${EDITOR:-vi}}** *count* in the input buffer. If *count* is omitted, then the current line is used.
^L	Line feed and print current line. Has effect only in control mode.
^J	(New line) Execute the current line, regardless of mode.
^M	(Return) Execute the current line, regardless of mode.
#	Equivalent to **I#**<*cr*>. Useful for causing the current line to be inserted in the history without being executed.

Special Commands.
The following simple-commands are executed in the shell process. Input/Output redirection is permitted. File descriptor 1 is the default output location. Parameter assignment lists preceding the command do not remain in effect when the command completes unless noted.

: [*arg* ...]
Parameter assignments remain in effect after the command completes. The command only expands parameters. A zero exit code is returned.

. *file* **[** *arg* . . . **]**
Parameter assignments remain in effect after the command completes.

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

alias [**−tx**] [*name*[**=***value*] ...]
Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **−t** flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **−t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **−x** flag is used to set or print exported aliases. An exported alias is defined across sub-shell environments. Alias returns true unless a *name* is given for which no alias has been defined.

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

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

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

cd [*arg*]
cd *old new*
This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is − the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a **/** then the search path is not used. Otherwise, each directory in the path is searched for *arg*.
The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.
The **cd** command may not be executed by *rsh*.

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

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

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

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

fc [**−e** *ename* **] [−nlr**] [*first*] [*last*]
fc −e − [*old=new*] [*command*]
In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag **−l**, is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the parameter **FCEDIT** (default /bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed. *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and −16 for listing. The flag **−r** reverses the order of the commands and the flag **−n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old=new* is performed.

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

jobs [**−l**]
Lists the active jobs; given the **−l** options lists process id's in addition to the normal information.

kill [*−sig*] *process* . . .
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix ''SIG''). The signal names are listed by **kill −l**'. There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument *process* can be either a process id or a job.

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

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

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

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

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

newgrp [*arg* . . .]
Equivalent to **exec newgrp** *arg* . . . .

print [**−Rnprsu**[*n*]] [*arg* . . .]
The shell output mechanism. With no flags or with flag −, the arguments are printed on standard output as described by *echo*(1). In raw mode, **−R** or **−r**, the escape conventions of *echo* are ignored. The **−R** option will print all subsequent arguments and options other than **−n**. The **−p** option causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output. The **−s** option causes the arguments to be written onto the history file

instead of standard output. The **−u** flag can be used to specify a one digit file descriptor unit number **n** on which the output will be placed. The default is 1. If the flag **−n** is used, no **new-line** is added to the output.

read [**−prsu**[*n*]] [*name?prompt*] [*name* ...]
The shell input mechanism. One line is read and is broken up into words using the characters in **IFS** as separators. In raw mode, **−r,** a \ at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The **−p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **−s** fag is present, the input will be saved as a command in the history file. The flag **−u** can be used to specify a one digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted then **REPLY** is used as the default *name*. The return code is 0 unless an end-of-file is encountered. An end-of-file with the **−p** option causes cleanup for this process so that another can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device then the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is 0 unless an end-of-file is encountered.

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

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

set [**−aefhkmnostuvx**] [**−o** *option* ...] [*arg* ...]
The flags for this command have meaning as follows:

−a	All subsequent parameters that are defined are automatically exported.
−e	If the shell is non-interactive and if a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.
−f	Disables file name generation.
−h	Each command whose name is an *identifier* becomes a tracked alias when first encountered.
−k	All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.
−m	Background jobs will run in a separate process group and a line will print upon completion. The exit status of

background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

−n Read commands but do not execute them.

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

allexport
Same as **−a**.

errexit Same as **−e**.

emacs Puts you in an *emacs* style in-line editor for command entry.

gmacs Puts you in a *gmacs* style in-line editor for command entry.

ignoreeof
The shell will not exit on end-of-file. The command **exit** must be used.

keyword Same as **−k**.

markdirs
All directory names resulting from file name generation have a trailing **/** appended.

monitor Same as **−m**.

noexec Same as **−n**.

noglob Same as **−f**.

nounset Same as **−u**.

verbose Same as **−v**.

trackall
Same as **−h**.

vi Puts you in insert mode of a *vi* style in-line editor until you hit escape character **033**. This puts you in move mode. A return sends the line.

viraw Each character is processed as it is typed in *vi* mode.

xtrace Same as **−x**.
If no option name is supplied then the current option settings are printed.

−s Sort the positional parameters.

−t Exit after reading and executing one command.

−u Treat unset parameters as an error when substituting.

−v Print shell input lines as they are read.

−x Print commands and their arguments as they are executed.

− Turns off **−x** and **−v** flags and stops examining arguments for flags.

—— Do not change any of the flags; useful in setting **$1** to a value beginning with −. If no arguments follow this flag then the positional parameters are unset.

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

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

test [*expr*]
Evaluate conditional expression *expr*. See *test*(1) for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional primitive expressions are allowed:
−L *file*
True if *file* is a symbolic link.
file1 **−nt** *file2*
True if *file1* is newer than *file2*.
file1 **−ot** *file2*
True if *file1* is older than *file2*.
file1 **−ef** *file2*
True if *file1* has the same device and i-node number as *file2*.

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

trap [*arg*] [*sig*] ...
arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is omitted or is −, then all trap(s) *sig* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit code. This trap is not inherited by functions. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

typeset [**−FLRZefilprtux**[*n*] [*name*[*=value*]] ...]
Parameter assignments remain in effect after the command completes. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:
−F This flag provides UNIX to host-name file mapping on non-UNIX machines.

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

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

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

−e Tag the parameter as having an error. This tag is currently unused by the shell and can be set or cleared by the user.

−f The names refer to function names rather than parameter names. No assignments can be made and the only other valid flag is **−x**.

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

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

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

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

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

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

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

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

ulimit [**−cdfmpt**] [*n*]

−c imposes a size limit of *n* blocks on the size of core dumps (BSD only).

−d imposes a size limit of *n* blocks on the size of the data area (BSD only).

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

−m imposes a soft limit of *n* blocks on the size of physical memory (BSD only).

−p changes the pipe size to *n* (UNIX/RT only).
−t imposes a time limit of *n* seconds to be used by each process (BSD only).

If no option is given, **−f** is assumed. If *n* is not given the current limit is printed.

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

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

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

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

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

Invocation.
If the shell is invoked by *exec*(2), and the first character of argument zero (**$0**) is −, then the shell is assumed to be a *login* shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV** if the file exists. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

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

ignored by the shell.

−r If the **−r** flag is present the shell is a restricted shell.

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

Rsh Only.

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **SHELL** or **PATH,**
specifying path or command names containing **/**,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

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

SEE ALSO

cat(1), cd(1), echo(1), emacs(1), env(1), gmacs(1), newgrp(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), gtty(2), pipe(2), signal(2), umask(2), ulimit(2), wait(2), rand(3), a.out(5), profile(5), environ(7).

CAVEATS

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

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

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

Ksh enhancements by Goran Larsson

E001

25-jun-1988
Single filename expansion in EMACS/GMACS mode.
ESC ESC will fill in as much as is possible of
the filename. A beep indicates incomplete filename.

E002

26-jun-1988
Runaway limit. Quick repeat of CR caused garbage
to appear due to switching raw/cooked. Fixed.

E003

26-jun-1988
It is now possible to go below the last saved command
in the history list. In other words, you may go
back to the empty line that you had before you started
to climb the history list.

E004

26-jun-1988
The option quiet. "set -o quiet" mutes the beep produced
during single filename expansion (E001). "set +o quiet"
brings it back again.

E005

04-aug-1988
Builtin "exit" is illegal for login shell.
Builtin "logout" is used to logout from login shell.
Reversed for children.

E006

04-aug-1988
The builtin command "jobs" returns number of jobs
as it's exit status.

E007

04-aug-1988
/etc/ksh/profile and ~/.profile executed at login.
/etc/ksh/unprofile and ~/.unprofile executed at logout.

E008

27-aug-1988
The builtin variable TTY is set to the terminal name
when the shell starts. The TTY variable is not the same
as the terminal name returned by ttyname() just after
logging in. The following example shows the difference
in a window system:

```
login tty = console     TTY = wtty01     for first window
login tty = console     TTY = wtty02     for second window
login tty = console     TTY = wtty03     for third window
```

The history file name is now built by the following
algorithm:

```
if HISTFILE is undefined then
        name := HOME + "/.hist_" + TTY
else
        name := HISTFILE + TTY
```

endif

This change prevents history files from being corrupted when ksh is started from a window system or an user is logging in on several terminals at the same time.

E009 27-aug-1988
Named parameters (variables) and functions that has a name that starts with two underscores are not displayed by the 'set' and 'typeset' builtins. This makes it possible to have hidden global variables in shell procedures.

```
KSH/
KSH/etc/
KSH/etc/ksh/
KSH/etc/ksh/dirstack
KSH/etc/ksh/environment
KSH/etc/ksh/profile
KSH/etc/ksh/timezone
KSH/etc/ksh/unprofile
KSH/etc/sh/
KSH/etc/sh/profile
KSH/etc/sh/timezone
KSH/etc/csh/
KSH/etc/csh/login
KSH/etc/csh/logout
KSH/etc/csh/timezone
KSH/root/
KSH/root/:.profile
KSH/root/:.environment
KSH/anybody/
KSH/anybody/:.environment
KSH/anybody/:.profile
KSH/bin/
KSH/bin/ksh
```