Bell Laboratories

UNPL
1404

subject: The PWB/UNIX Edition 1.2.1 Bourne Shell
Case: 70149-002
File: 39382-900

date: May 2, 1979

from: M. J. Fitton
PY 9442
2G-213 x6782

MEMORANDUM FOR FILE

1. Introduction

This document describes the differences between the PWB/UNIX™ Edition 1.2 Bourne Shell and the Edition 1.2.1 Bourne Shell. Most differences are upward compatible with the exception of variables that are set by the Shell. Attached is the manual page describing this version of the Shell, and the manual page describing the Shell's built-in test program.

2. Changes

This section describes the Edition 1.2.1 changes. These changes are reflected in the attached bsh(I) manual page.

2.1 Shell Variables

1. $n is no longer available, use $#.
2. $p is no longer available, use $PATH.
3. $r is no longer available, use $?.
4. $pid is no longer available, use $$.
5. $pcs is no longer available, use $!.
6. $h is no longer available, use $HOME.

2.2 Shell Performance

1. Test (alias "[") has been built into the Shell.
2. The Shell no longer forks on variable assignment.
3. test and "[" are equivalent with respect to performance (previously, "[" took longer).

2.3 Enhancements

1. New "+" option added to set:

   ```
   set -k  -> sets the k flag
   set +k  -> unsets the k flag
   ```

   This option applies to the following flags:
   x, k, t, n, e, v

2. New strip-leading-tabs option on "here-documents". To strip leading tabs use "<<-" rather than "<<".

   cat <<-! will strip leading tabs up to and including the "eof" character (!).

3. New (upward compatible) test program built into the Shell. See Attachment 2.

2.4 Other Changes

1. The $PATH variable will be set from the .path file and automatically "exported". This will not be the case in Release 2.0; users will have to set and export $PATH from their .profile.

2. Login is no longer recognized by bsh. In Release 2.0, login is no longer executable by users.

3. The Shell will not allow you to trap signal 11. This is because of its internal use of signal 11. The only signals that should be trapped by Shell procedures are 0, 1, 2, 3 and 15.

M. J. Fitton

PY-9442-MJF-mjf

Copy (without att.) to
A. P. Boysen, Jr.
M. P. Fabisch
Supervision, Department 9442

NAME

bsh - shell, the standard command programming language

SYNOPSIS

bsh [-ceiknrstuvx] [arg] ...

DESCRIPTION

Bsh is a command programming language that executes commands read from a terminal or a file. See Invocation below for the meaning of arguments to the shell.

Commands.

A simple-command is a sequence of non-blank words separated by blanks (a blank is a tab or a space). 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.

A list is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four 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 the list following it to be executed only if the preceding pipeline returns a zero (non-zero) value. 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. The value returned by a command is that of the last simple-command executed in the command.

for name [in word ...] do list done

Each time a for command is executed, name is set to the next word taken from the in word list. If in word ... is omitted, then in "$@" is assumed. Execution ends when there are no more words in the list.

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.

if list then list [elif list then list] ... [else list] fi
The list following if is executed and, if it returns zero, the list following then is executed. Otherwise, the list following elif is executed and, if its value is zero, the list following then is executed. Failing that, the else list is executed.

while list do list done
A while command repeatedly executes the while list and, if its value is zero, executes the do list; otherwise the loop terminates. The value returned by a while command is that of the last executed command in the do list; until may be used in place of while to negate the loop termination test.

(list)
Execute list in a sub-shell.

{list;}
list is simply executed.

The following words 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 { }

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.

Parameter Substitution.
The character $ is used to introduce substitutable parameters. Positional parameters may be assigned values by set. Variables may be set by writing:

name=value [name=value] ...

${parameter}
A parameter is a sequence of letters, digits, or underscores (a name), a digit, or any of the characters *, @, #, ?, -, $, and !. The value, if any, of the parameter is substituted. The braces are required only when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A name must begin with a letter or underscore. 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). Parameter $0 is set from argument zero when the shell is invoked.

${parameter-word}
If parameter is set then substitute its value; otherwise substitute word.

${parameter=word}
If parameter is not set 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 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 then substitute word; otherwise substitute nothing.

In the above word is not evaluated unless it is to be used as the substituted string, so that, for example:

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

will only execute pwd if d is unset.

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 process number of the last background command invoked.

The following parameters are used by the shell:

HOME	The default argument (home directory) for the cd command.
PATH	The search path for commands (see Execution below).
MAIL	If this variable is set to the name of a mail file then the shell informs the user of the arrival of mail in the specified file.
PS1	Primary prompt string, by default $.
PS2	Secondary prompt string, by default >.
IFS	Internal field separators, normally space, tab, and new-line.

The shell gives default values to PATH, PS1, PS2, and IFS, while HOME and MAIL are not set at all by the shell

(although HOME is set by login(1)).

Blank Interpretation.
After parameter and command substitution, the results of substitution are scanned for internal 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 [. 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. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

* 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.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted:

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

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" ....

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. Substitution occurs before word or digit is used.

<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. The resulting 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 >.

<&- 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 created is that specified by the digit (instead of the default 0 or 1). For example:

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

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by & 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 shell interacts with the en-

vironment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the export command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in export commands.

The environment for any simple-command may be augmented by prefixing it with one or more assignments to parameters. 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 keyword arguments are placed in the environment, even if they occur after the command name. The following prints `a=b c' and `c':

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

Signals.
The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; 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. Except for the Special Commands listed below, a new 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). 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 an a.out file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A

parenthesized command is also executed in a sub-shell.

Special Commands.
The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands.

: No effect; the command does nothing.

.. file
Read and execute commands from file and return. The search path specified by PATH is used to find the directory containing file.

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

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

cd [arg]
Change the current directory to arg. The shell parameter HOME is the default arg.

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

exec [arg ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [n]
Causes a non-interactive 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.)

export [name ...]
The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given, then a list of all name-value pairs in the environment is printed.

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

read [name ...]
One line is read from the standard input and 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 return code is 0 unless an end-of-file is encountered.

readonly [name ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, then a list of all readonly names is printed.

set [-ekntuvx [arg ...]]
-e If the shell is non-interactive then exit immediately if a command fails.
-k All keyword arguments are placed in the environment for a command, not just those that precede the command name.
-n Read commands but do not execute them. — doesn't really check all syntax
-t Exit after reading and executing one command.
-u Treat unset variables as an error when substituting.
-v Print shell input lines as they are read.
-x Print commands and their arguments as they are executed.

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 $-. 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.

shift
The positional parameters from $2 ... are renamed $1 ....

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

trap [arg] [n] ...
arg is a command to be read and executed when the shell receives signal(s) n. (Note that arg is scanned once when the trap is set and once when the trap is taken.) 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 absent then all trap(s) n 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 n is 0 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.

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.

Invocation.
If the first character of argument zero ($0) is -, commands are read from /etc/profile and then from $HOME/.profile, if such files exist. 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 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 interruptable). In all cases, QUIT is ignored by the shell.
-r If the -r flag is present the shell is a restricted shell (see rsh(1)).

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

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

SEE ALSO

login(1), rsh(1), test(1), dup(2), exec(2), fork(2), gtty(2), pipe(2), signal(2), wait(2), a.out(5), profile(5), environ(7).

BUGS

The command readonly (without arguments) produces the same output as the command export.
If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created and the shell complains about not being able to find that file by another name.

NAME

test - condition evaluation command

SYNOPSIS

test expr
- or -
[expr]

DESCRIPTION

Test evaluates the expression expr and, if its value is true, returns zero (true) exit status; otherwise, a non-zero (false) exit status is returned; test returns a non-zero exit if there are no arguments.

The following primitives are used to construct expr:

-r file	true if file exists and is readable.
-w file	true if file exists and is writable.
-x file	true if file exists and is executable.
-f file	true if file exists and is a regular file.
-d file	true if file exists and is a directory.
-c file	true if file exists and is a character special file.
-b file	true if file exists and is a block special file.
-u file	true if file exists and its set user ID bit is set.
-g file	true if file exists and its set group ID bit is set.
-k file	true if file exists and its sticky bit is set.
-s file	true if file exists and has a size greater than zero.
-t [fildes]	true if the open file whose file descriptor number is fildes (1 by default) is associated with a terminal device.
-z s1	true if the length of string s1 is zero.
-n s1	true if the length of the string s1 is nonzero.
s1 = s2	true if strings s1 and s2 are equal.
s1 != s2	true if strings s1 and s2 are not equal.

s1 true if s1 is not the null string.

n1 -eq n2 true if the integers n1 and n2 are algebraically equal. Any of the comparisons -ne, -gt, -ge, -lt, and -le may be used in place of -eq.

These primaries may be combined with the following operators:

! unary negation operator

-a binary and operator

-o binary or operator

(expr) parentheses for grouping.

-a has higher precedence than -o.

Notice that all the operators and flags are separate arguments to test. Notice also that parentheses are meaningful to the Shell and, therefore, must be escaped.

SEE ALSO
sh(1), find(1)

NAME

shcvt - Shell conversion aid

SYNOPSIS

shcvt file

DESCRIPTION

Shcvt performs partial conversion of PWB 1.0 Shell files into UNIX/TS Shell files. A partially converted file xxx.b (where xxx is the name of the file that was converted) is created in the current directory.

Further conversion by hand is generally required unless the Shell file is a very simple one.

FILES

/usr/lib/clean
/usr/lib/sconvert
/usr/lib/script1
/usr/lib/fix

SEE ALSO

Shcvt - A Tool For Shell Conversion by G. A. Snyder
sh(I)

BUGS

The input must be a syntactically correct PWB 1.0 Shell file.
Behavior is unpredictable on some parenthesized expressions.

NAME

idcvt - convert SCCS User IDs to login names

SYNOPSIS

idcvt file ...

DESCRIPTION

Idcvt assumes that any number found in the User List of an SCCS file (see sccsfile(V)) is a numerical User ID and replaces it with the login name (or names) associated with that User ID.

Arguments to idcvt can only be SCCS file names. Idcvt carries out its function by executing the appropriate admin(I) command. Files that do not require conversion are silently ignored, whereas names of files that are undergoing conversion are printed on the standard output.

Since User IDs in the User List of an SCCS file are not supported by SCCS, this command must be run prior to installation of Release 2.0 of PWB/UNIX.

The user should time conversion properly since all numbers in an SCCS file User List after the installation of Release 2.0 of PWB/UNIX will be interpreted as Group IDs by SCCS commands.

SEE ALSO

admin(I), prs(I), help(I), sccsfile(V).

DIAGNOSTICS

Use help(I) for explanations.