## NAME

ush – the micro (mu) shell command interpreter.

## SYNOPSIS

**ush**

## DESCRIPTION

*ush* is a command interpreter with a syntax similar to UNIX C shell, *csh*(1). However, it is for instructional purposes only, therefore it is much simpler.

### Initialization and Termination

When first stared, *ush* normally performs commands from the file *~/.ushrc*, provided that it is readable. Commands in this file are processed just the same as if they were taken from standard input.

### Interactive Operation

After startup processing, an interactive *ush* shell begins reading commands from the terminal, prompting with *hostname%*. The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*; this sequence of words is parsed (as described under **Usage**); and the shell executes each command in the current line.

## USAGE

### Lexical Structure

The shell splits input lines into words separated by whitespace (spaces or tabs), with the following exceptions:

> The special characters &, |, ;, <, and > and the multi-character sequences >>, |&, >& and >>& are always separate words, whether or not they are surrounded by whitespace.
>
> Special characters preceded by a backslash character (\) are not interpreted by the shell. Two backslashes together sends the backslash character to the shell; otherwise the backslash is stripped from the input.
>
> Characters enclosed in double quotes (") or single quotes (') form a single word. Special characters inside of strings do not form separate words.

### Command Line Parsing

A *simple command* is a sequence of words, the first of which specifies the command to be executed. A *pipeline* is a sequence of one or more simple commands separated by | or |&. With |, the standard output of the preceding command is redirected to the standard input of the command that follows. With |&, both the standard error and the standard output are redirected through the pipeline.

A *list* is a sequence of one or more pipelines separated by ; or &. Pipelines joined into sequences with ; will be executed sequentially. Pipelines ending with & are executed asynchronously. In which case, the shell does not wait for the pipeline to finish; instead, it displays the job number (see **Job Control**) and associated process ID, and begins processing the subsequent pipelines (prompting if necessary).

### I/O Redirection

The following separators indicate that the subsequent word is the name of a file to which the command's standard input, standard output, or standard error is redirected.

| | |
|---|---|
| < | Redirect the standard input. |
| >, >& | Redirect the standard output to a file. If file does not exist, it is created. If it does exist, it is overwritten and its previous contents are lost. The & form redirects both standard output and standard error to the file. |
| >>, >>& | Append the standard output. Like >, but places output at the end of the file rather than overwriting. The & form appends both standard output and standard error to the file. |

**Command Execution**

If the command is an *ush* shell built-in, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command-name contains a /, the shell takes it as a pathname and searches for it. If a pathname begins with a /, then the path is absolute; otherwise, the path is relative to the current working directory. If the command-name does not contain a /, the shell attempts to resolve it to a pathname, searching each directory in the PATH variable for the command.

When a file, with its pathname resolved, is found that has proper execute permission, the shell forks a new process to run the command. The shell also passes along any arguments to the command. Using one of the flavor of *exec* system call, such as *execv*(2V), the newly forked process attempts to overlay the desired program. If successful, the shell is silent.

If the file does not have execute permissions, or if the pathname matches a directory, a ‘‘permission denied’’ message is displayed. If the pathname cannot be resolved a ‘‘command not found’’ message is displayed. If either of these errors occurs with any component of a pipeline the entire pipeline is aborted, even though some of the pipeline may already be executing.

**Environment Variables**

Environment variables may be accessed via the **setenv** built-in commands. When a program is exec’d the environment variables are passed as parameters to *execv* or equivalent.

**Signal Handling**

The shell ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including hangups (HUP). Other signals have the values that *ush* inherited from its environment. *Ush* catches the TERM signal.

**Job Control**

The shell associates a numbered *job* with each command sequence, to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically CTRL-Z). When a command is started in the background using &, the shell displays a line with the job number in brackets and the process number; e.g., [1] 2345

To see the current list of jobs, use the **jobs** built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job.

To manipulate jobs, refer to the built-in commands **bg**, **fg**, **kill**.

A reference number to a job begins with a ‘%’. To refer to job number *j* use *%j*, as in: *bg %j*.

**Status Reporting**

While running interactively, the shell tracks the status of each job and reports whenever it finishes or becomes blocked. It displays a message to this effect as it issues a prompt, so as to avoid disturbing the appearance of your input.

**Builtin commands**

Built-in commands are executed within *ush*. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

**bg** *%job*
: Puts the specified job into the background.

**cd** [*dir*]
: Change the working directory of the shell to *dir*, provided it is a directory and the shell has the appropriate permissions. Without an argument, it changes the working directory to the original (home) directory.

**fg** *%job*
: Brings the specified job into the foreground.

**echo** *%word ...*
: Writes each *word* to the shell’s standard output, separated by spaces and terminated with a newline.

**jobs**
: Lists the active jobs.

**kill** %*job*
Send the TERM (terminate) signal to the indicated job.

**logout** Exits the shell.

**nice** [**[+/-]***number*] [*command*]
Sets the scheduling priority for the shell to *number*, or, without *number*, to 4. With *command*, runs *command* at the appropriate priority. The greater the *number*, the less cpu the process gets. If no sign before the number, assume it is positive.

**pwd** Prints the current working directory.

**setenv** [*VAR* [*word*]]
Without arguments, prints the names and values of all environment variables. Given *VAR*, sets the environment variable *VAR* to *word* or, without *word*, to the null string.

**unsetenv** *VAR*
Removes environment variable whose name matches *VAR*.

**where** *command*
Reports all known instances of *command*, including builtins and executables in **path**.

## FILES

| | |
|---|---|
| *~/.ushrc* | Read at the beginning of execution by each shell. |

## SEE ALSO

csh(1), dup(2), execv(2), fork(2), killpg(2), pipe(2), sigvec(2), vfork(2), wait(2), environ(7), Introduction to the C Shell