NAME

while, until – shell built-in functions to repetitively execute a set of actions while/until conditions are evaluated TRUE

SYNOPSIS

sh

while [*conditions* **]; do** *actions* **; done**
until [*conditions* **]; do** *actions* **; done**

csh

while (*conditions***)**
. . . *# do actions*
end

ksh

while [*conditions* **]; do** *actions* **; done**
until [*conditions* **]; do** *actions* **; done**

DESCRIPTION

sh

A **while** command repeatedly executes the **while** *conditions* and, if the exit status of the last command in the *conditions* list is **0**, executes the **do** *actions*; otherwise the loop terminates. If no commands in the **do** *actions* are executed, then the **while** command returns a **0** exit status; **until** may be used in place of **while** to negate the loop termination test.

csh

While *conditions* is **TRUE** (evaluates to nonzero), repeat commands between the **while** and the matching **end** statement. The **while** and **end** must appear alone on their input lines. If the shell's input is a terminal, it prompts for commands with a question-mark until the **end** command is entered and then performs the commands in the loop.

ksh

A **while** command repeatedly executes the **while** *conditions* and, if the exit status of the last command in the *conditions* list is zero, executes the **do** *actions*; otherwise the loop terminates. If no commands in the **do** *actions* are executed, then the **while** command returns a **0** exit status; **until** may be used in place of **while** to negate the loop termination test.

loop interrupts

The built-in command **continue** may be used to terminate the execution of the current iteration of a **while** or **until** loop, and the built-in command **break** may be used to terminate execution of a **while** or **until** command.

EXAMPLES

In these examples, the user is repeated prompted for a name of a file to be located, until the user chooses to finish the execution by entering an empty line.

sh

```
filename=anything
while [ $filename ]
do
        echo "file?"
        read filename           # read from terminal
        find . -name $filename -print
done
```

The brackets surrounding **$filename** are necessary for evaluation. (See the **test** built-in command in the **if**(1) man page). Additionally, there must be a blank space separating each bracket from any characters within.

csh

```
set filename = anything
while ( "$filename" != "" )
        echo "file?"
        set filename = $<                # read from terminal
        find . -name $filename -print
end
```

ksh

Use the same syntax as in the Bourne shell, **sh**, example above.

SEE ALSO

break(1), **csh**(1), **if**(1), **ksh**(1), **sh**(1)

NOTES

Both the Bourne shell, **sh**, and the Korn shell, **ksh**, can use the semicolon and the carriage return interchangeably in their syntax of the **if**, **for**, and **while** built-in commands.