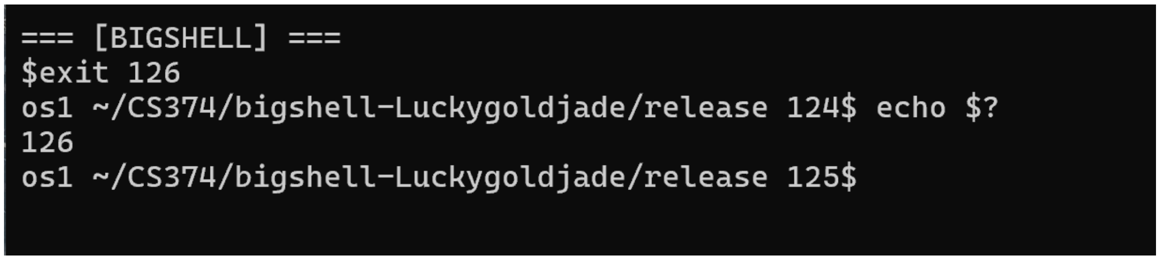
BigShell Annotated Test Examples

Test 1: exit Built-in – Preserve Status

This test ensures the BigShell correctly exits with the provided status code, and that the parent shell captures that status with `$?`.

```
$ exit 126      # Exit the shell with code 126
$ echo $?       # Should print 126
```

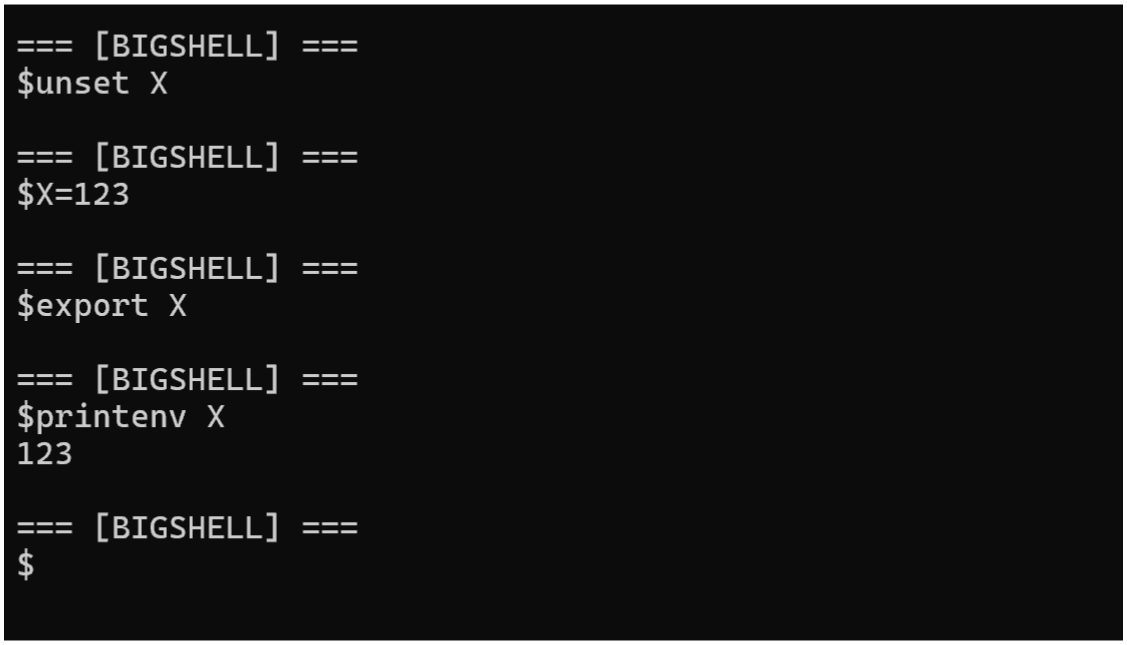


```
=== [BIGSHELL] ===
$exit 126
os1 ~/CS374/bigshell-Luckygoldjade/release 124$ echo $?
126
os1 ~/CS374/bigshell-Luckygoldjade/release 125$
```

Test 2: export and printenv – Environment Passing

This verifies that exported variables are available to child processes.

```
$ unset X                     # Ensure X is not set
$ X=123                       # Set variable X internally
$ export X                    # Export X so child processes can see it
$ printenv X                  # Should print '123'
```

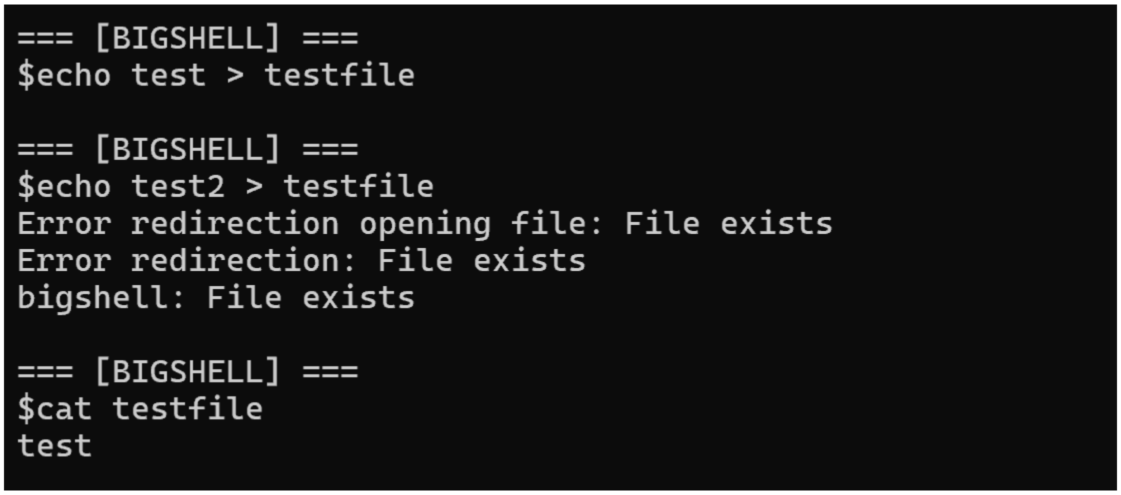


```
=== [BIGSHELL] ===
$unset X

=== [BIGSHELL] ===
$X=123

=== [BIGSHELL] ===
$export X

=== [BIGSHELL] ===
$printenv X
123

=== [BIGSHELL] ===
$
```

Test 3: > Redirection – File Protection

This test confirms that redirection using `>` will not overwrite an existing file unless allowed.

```
$ echo test > testfile              # Create a file with content 'test'
$ echo test2 > testfile             # Try to overwrite it (should fail

                                    or be blocked)
$ cat testfile                      # Should still show 'test'
```

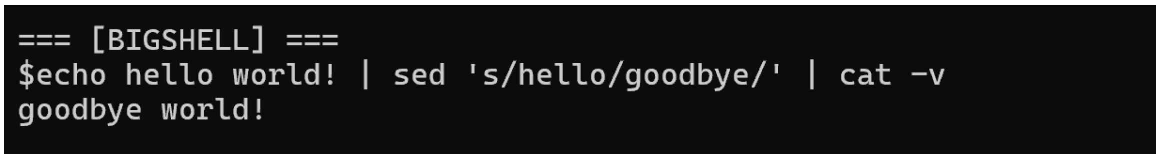


```
=== [BIGSHELL] ===
$echo test > testfile

=== [BIGSHELL] ===
$echo test2 > testfile
Error redirection opening file: File exists
Error redirection: File exists
bigshell: File exists

=== [BIGSHELL] ===
$cat testfile
test
```

Test 4: echo | sed | cat – Pipelining

This demonstrates proper support for pipelines in BigShell, chaining commands together.

```
$ echo hello world! | sed 's/hello/goodbye/' | cat -v
# Should output: goodbye world!
```

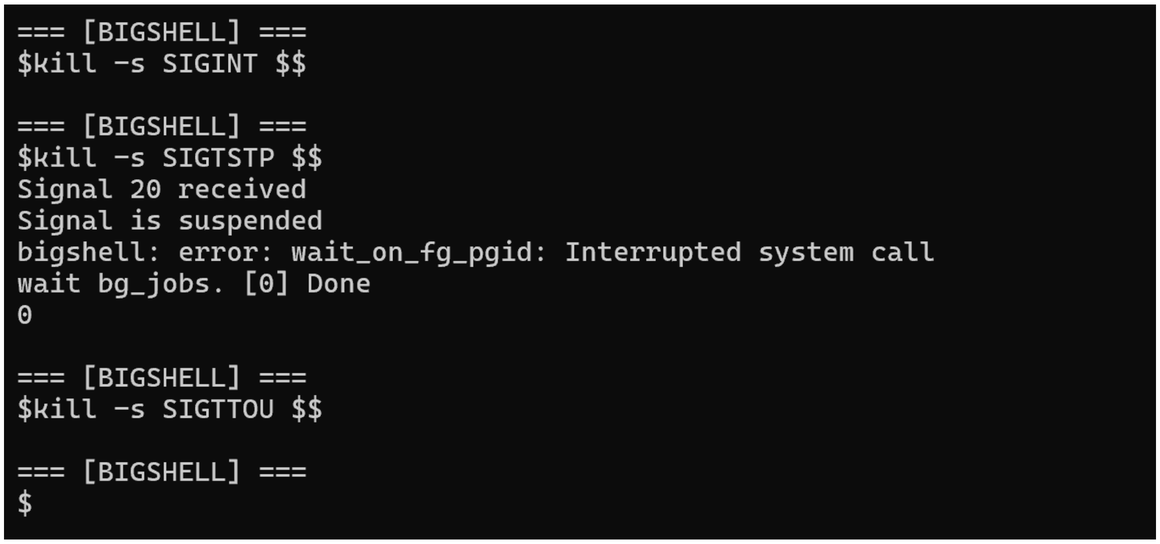


```
=== [BIGSHELL] ===
$echo hello world! | sed 's/hello/goodbye/' | cat -v
goodbye world!
```

Test 5: Signal Handling – Shell Ignores Ctrl Signals

Test whether BigShell ignores signals such as SIGINT (Ctrl-C), SIGTSTP (Ctrl-Z), and SIGTTOU.

```
$ kill -s SIGINT $$      # Send SIGINT (Ctrl-C) to the shell itself

--


$ kill -s SIGTSTP $$     # Send SIGTSTP (Ctrl-Z)

Signal 20 received       # SIGTSTP signal is caught

Signal is suspended      # My custom signal handler confirms shell

                         suspension

bigshell: error: wait_on_fg_pgid: Interrupted system call

                         # waitpid() was interrupted by the signal, as

                         expected

wait bg_jobs. [0] Done   # Background job has completed

0                        # $? correctly shows exit status of the last

                         successful command


--

$ kill -s SIGTTOU $$     # Send SIGTTOU (write to background process)
```

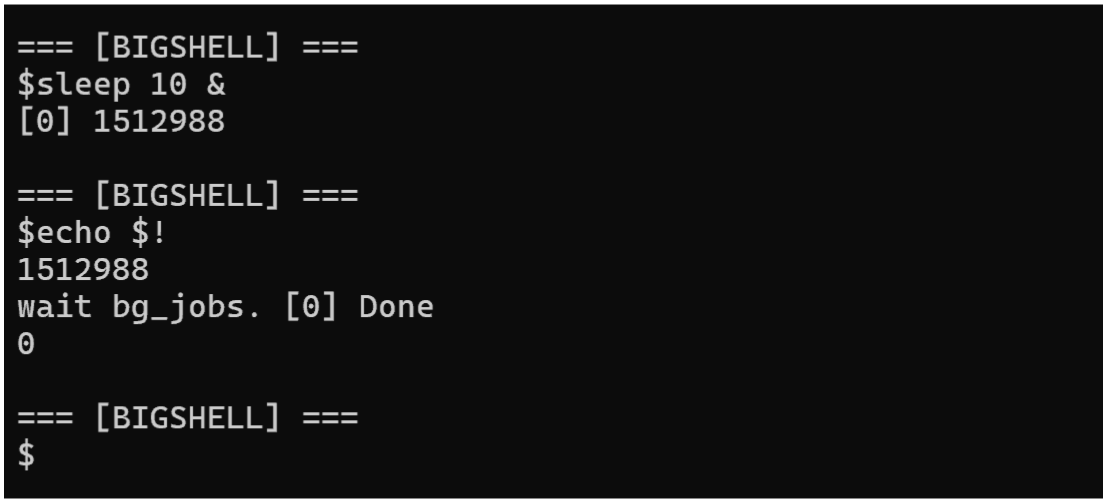


```
=== [BIGSHELL] ===
$kill -s SIGINT $$

=== [BIGSHELL] ===
$kill -s SIGTSTP $$
Signal 20 received
Signal is suspended
bigshell: error: wait_on_fg_pgid: Interrupted system call
wait bg_jobs. [0] Done
0

=== [BIGSHELL] ===
$kill -s SIGTTOU $$

=== [BIGSHELL] ===
$
```

Test 6: Background Process – $! Shows PID

Test confirms that BigShell properly updates the special parameter `$!` with the last background job's PID.

```
$ sleep 10 &             # Launch a background job
[0] 12345                # Sample output with job ID and PID
$ echo $!                # Should print the PID of the last background

12345                     process (e.g., 12345)
```



```
=== [BIGSHELL] ===
$sleep 10 &
[0] 1512988

=== [BIGSHELL] ===
$echo $!
1512988
wait bg_jobs. [0] Done
0

=== [BIGSHELL] ===
$
```