

Chapter 16. I/O Redirection

Table of Contents

- 16.1. [Using exec](#)
- 16.2. [Redirecting Code Blocks](#)
- 16.3. [Applications](#)

There are always three default "files" open, `stdin` (the keyboard), `stdout` (the screen), and `stderr` (error messages output to the screen). These, and any other open files, can be redirected. Redirection simply means capturing output from a file, command, program, script, or even code block within a script (see [Example 3-1](#) and [Example 3-2](#)) and sending it as input to another file, command, program, or script.

Each open file gets assigned a file descriptor. [1] The file descriptors for `stdin`, `stdout`, and `stderr` are 0, 1, and 2, respectively. For opening additional files, there remain descriptors 3 to 9. It is sometimes useful to assign one of these additional file descriptors to `stdin`, `stdout`, or `stderr` as a temporary duplicate link. [2] This simplifies restoration to normal after complex redirection and reshuffling (see [Example 16-1](#)).

```
COMMAND_OUTPUT >
# Redirect stdout to a file.
# Creates the file if not present, otherwise overwrites it.

ls -lR > dir-tree.list
# Creates a file containing a listing of the directory tree.

: > filename
# The > truncates file "filename" to zero length.
# If file not present, creates zero-length file (same effect as 'touch').
# The : serves as a dummy placeholder, producing no output.

> filename
# The > truncates file "filename" to zero length.
# If file not present, creates zero-length file (same effect as 'touch').
# (Same result as ": >", above, but this does not work with some shells.)

COMMAND_OUTPUT >>
# Redirect stdout to a file.
# Creates the file if not present, otherwise appends to it.

# Single-line redirection commands (affect only the line they are on):
# -----

1>filename
# Redirect stdout to file "filename".
1>>filename
# Redirect and append stdout to file "filename".
```

```

2>filename
# Redirect stderr to file "filename".
2>>filename
# Redirect and append stderr to file "filename".
&>filename
# Redirect both stdout and stderr to file "filename".

#=====
# Redirecting stdout, one line at a time.
LOGFILE=script.log

echo "This statement is sent to the log file, \"\$LOGFILE\"." 1>\$LOGFILE
echo "This statement is appended to \"\$LOGFILE\"." 1>>\$LOGFILE
echo "This statement is also appended to \"\$LOGFILE\"." 1>>\$LOGFILE
echo "This statement is echoed to stdout, and will not appear in \"\$LOGFILE\"."
# These redirection commands automatically "reset" after each line.

# Redirecting stderr, one line at a time.
ERRORFILE=script.errors

bad_command1 2>\$ERRORFILE      # Error message sent to $ERRORFILE.
bad_command2 2>>\$ERRORFILE     # Error message appended to $ERRORFILE.
bad_command3                   # Error message echoed to stderr,
                                #+ and does not appear in $ERRORFILE.
# These redirection commands also automatically "reset" after each line.
#=====

2>&1
# Redirects stderr to stdout.
# Error messages get sent to same place as standard output.

i>&j
# Redirects file descriptor i to j.
# All output of file pointed to by i gets sent to file pointed to by j.

>&j
# Redirects, by default, file descriptor 1 (stdout) to j.
# All stdout gets sent to file pointed to by j.

0< FILENAME
< FILENAME
# Accept input from a file.
# Companion command to ">", and often used in combination with it.
#
# grep search-word <filename

[j]<>filename
# Open file "filename" for reading and writing, and assign file descriptor "j"

```

```

to it.
# If "filename" does not exist, create it.
# If file descriptor "j" is not specified, default to fd 0, stdin.
#
# An application of this is writing at a specified place in a file.
echo 1234567890 > File      # Write string to "File".
exec 3<> File              # Open "File" and assign fd 3 to it.
read -n 4 <&3              # Read only 4 characters.
echo -n . >&3              # Write a decimal point there.
exec 3>&-                  # Close fd 3.
cat File                  # ==> 1234.67890
# Random access, by golly.

|
# Pipe.
# General purpose process and command chaining tool.
# Similar to ">", but more general in effect.
# Useful for chaining commands, scripts, files, and programs together.
cat *.txt | sort | uniq > result-file
# Sorts the output of all the .txt files and deletes duplicate lines,
# finally saves results to "result-file".

```

Multiple instances of input and output redirection and/or pipes can be combined in a single command line.

```

command < input-file > output-file

command1 | command2 | command3 > output-file

```

See [Example 12-28](#) and [Example A-15](#).

Multiple output streams may be redirected to one file.

```

ls -yz >> command.log 2>&1
# Capture result of illegal options "yz" in file "command.log."
# Because stderr is redirected to the file,
#+ any error messages will also be there.

# Note, however, that the following does *not* give the same result.
ls -yz 2>&1 >> command.log
# Outputs an error message and does not write to file.

# If redirecting both stdout and stderr,
#+ the order of the commands makes a difference.

```

Closing File Descriptors

`n<&-`

Close input file descriptor *n*.

`0<&-`, `<&-`

Close `stdin`.

`n>&-`

Close output file descriptor *n*.

`1>&-`, `>&-`

Close `stdout`.

Child processes inherit open file descriptors. This is why pipes work. To prevent an fd from being inherited, close it.

```
# Redirecting only stderr to a pipe.

exec 3>&1                                # Save current "value" of stdout.
ls -l 2>&1 >&3 3>&- | grep bad 3>&-      # Close fd 3 for 'grep' (but not 'ls').
#           ^^^^      ^^^^
exec 3>&-                                # Now close it for the remainder of the
script.

# Thanks, S.C.
```

For a more detailed introduction to I/O redirection see [Appendix E](#).

Notes

- [1] A *file descriptor* is simply a number that the operating system assigns to an open file to keep track of it. Consider it a simplified version of a file pointer. It is analogous to a *file handle* in C.
- [2] Using *file descriptor 5* might cause problems. When Bash creates a child process, as with [exec](#), the child inherits fd 5 (see Chet Ramey's archived e-mail, [SUBJECT: RE: File descriptor 5 is held open](#)). Best leave this particular fd alone.

[Prev](#) Arithmetic Expansion

[Home](#)
[Up](#)

[Next](#)
Using `exec`