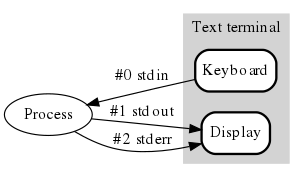
**Redirection (computing)**

**Redirection (computing)**

[](https://en.wikipedia.org/wiki/File:Stdstreams-notitle.svg)

The standard streams for input, output, and error

In [computing](https://en.wikipedia.org/wiki/Computing), **redirection** is a form of [interprocess communication](https://en.wikipedia.org/wiki/Interprocess_communication), and is a function common to most [command-line interpreters](https://en.wikipedia.org/wiki/Command-line_interpreter), including the various [Unix shells](https://en.wikipedia.org/wiki/Unix_shell) that can redirect [standard streams](https://en.wikipedia.org/wiki/Standard_streams) to user-specified locations.

In [Unix-like](https://en.wikipedia.org/wiki/Unix-like) operating systems, programs do redirection with the [dup2](https://en.wikipedia.org/wiki/Dup2)(2) [system call](https://en.wikipedia.org/wiki/System_call), or its less-flexible but higher-level [stdio](https://en.wikipedia.org/wiki/Standard_streams)analogues, [freopen](https://en.wikipedia.org/wiki/Freopen)(3) and [popen](https://en.wikipedia.org/w/index.php?title=Popen&action=edit&redlink=1)(3).

**Contents**

* [1Redirecting standard input and standard output](https://en.wikipedia.org/wiki/Redirection_(computing)#Redirecting_standard_input_and_standard_output)
  + [1.1Basic](https://en.wikipedia.org/wiki/Redirection_(computing)#Basic)
  + [1.2Variants](https://en.wikipedia.org/wiki/Redirection_(computing)#Variants)
* [2Piping](https://en.wikipedia.org/wiki/Redirection_(computing)#Piping)
* [3Redirecting to and from the standard file handles](https://en.wikipedia.org/wiki/Redirection_(computing)#Redirecting_to_and_from_the_standard_file_handles)
* [4Chained pipelines](https://en.wikipedia.org/wiki/Redirection_(computing)#Chained_pipelines)
* [5Redirect to multiple outputs](https://en.wikipedia.org/wiki/Redirection_(computing)#Redirect_to_multiple_outputs)
* [6See also](https://en.wikipedia.org/wiki/Redirection_(computing)#See_also)
* [7External links](https://en.wikipedia.org/wiki/Redirection_(computing)#External_links)

**Redirecting standard input and standard output**

Redirection is usually implemented by placing certain [characters](https://en.wikipedia.org/wiki/Token_(parser)) between [commands](https://en.wikipedia.org/wiki/Command_(computing)).

**Basic**

Typically, the [syntax](https://en.wikipedia.org/wiki/Syntax) of these characters is as follows, using < to redirect input, and > to redirect output.

command1 > file1

executes command1, placing the output in file1, as opposed to displaying it at the terminal, which is the usual destination for standard output. This will [clobber](https://en.wikipedia.org/wiki/Clobbering) any existing data in file1.

Using

command1 < file1

executes command1, with file1 as the source of input, as opposed to the [keyboard](https://en.wikipedia.org/wiki/Computer_keyboard), which is the usual source for standard input.

command1 < infile > outfile

combines the two capabilities: command1 reads from infile and writes to outfile

**Variants**

To append output to the end of the file, rather than clobbering it, the >> operator is used:

command1 >> file1

To read from a stream literal (an inline file, passed to the standard input), one can use a [here document](https://en.wikipedia.org/wiki/Here_document), using the << operator:

tr a-z A-Z << END\_TEXT

one two three

uno dos tres

END\_TEXT

To read from a string, one can use a [here string](https://en.wikipedia.org/wiki/Here_string), using the <<< operator:

tr a-z A-Z <<< "one two three"

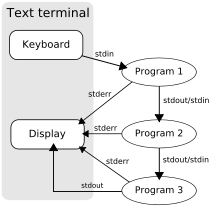
or:

NUMBERS="one two three"

tr a-z A-Z <<< "$NUMBERS"

**Piping**

Programs can be run together such that one program reads the output from another with no need for an explicit intermediate file:

[](https://en.wikipedia.org/wiki/File:Pipeline.svg)

A pipeline of three programs run on a text terminal

command1 | command2

executes command1, using its output as the input for command2 (commonly called [piping](https://en.wikipedia.org/wiki/Pipeline_(Unix)), with the "|" character being known as "pipe").

The two programs performing the commands may run in parallel with the only storage space being working buffers (Linux allows up to 64K for each buffer) plus whatever work space each command's processing requires. For example, a "sort" command is unable to produce any output until all input records have been read, as the very last record received just might turn out to be first in sorted order. Dr. Alexia Massalin's experimental operating system, [Synthesis](https://en.wikipedia.org/wiki/Synthesis_kernel#Massalin.27s_Synthesis_kernel), would adjust the priority of each task as they ran according to the fullness of their input and output buffers.

This produces the same end result as using two redirects and a temporary file, as in:

command1 > tempfile

command2 < tempfile

rm tempfile

But here, command2 does not start executing until command1 has finished, and a sufficiently large scratch file is required to hold the intermediate results as well as whatever work space each task required. As an example, although DOS allows the "pipe" syntax, it employs this second approach. Thus, suppose some long-running program "Worker" produces various messages as it works, and that a second program, TimeStamp copies each record from *stdin* to *stdout*, prefixed by the system's date and time when the record is received. A sequence such as

Worker | TimeStamp > LogFile.txt

Would produce timestamps only when Worker had finished, merely showing how swiftly its output file could be read and written.

A good example for command piping is combining [echo](https://en.wikipedia.org/wiki/Echo_(command)) with another command to achieve something interactive in a non-interactive shell, e.g.

echo -e 'user\npass' | ftp localhost

This runs the [ftp](https://en.wikipedia.org/wiki/File_Transfer_Protocol) client with input user, press return, then pass.

In casual use, the initial step of a pipeline is often cat or echo, reading from a file or string. This can often be replaced by input indirection or a [here string](https://en.wikipedia.org/wiki/Here_string), and use of cat and piping rather than input redirection is known as [useless use of cat](https://en.wikipedia.org/wiki/Useless_use_of_cat). For example, the following commands:

cat infile | cmd

echo $string | cmd

echo -e 'user\npass' | ftp localhost

can be replaced by:

cmd < infile

cmd <<< $string

ftp localhost <<< $'user\npass'

As echo is often a shell-internal command, its use is not as criticized as cat, which is an external command.

**Redirecting to and from the standard file handles**

In Unix shells derived from the original Bourneshell, the first two actions can be further modified by placing a number (the [file descriptor](https://en.wikipedia.org/wiki/File_descriptor)) immediately before the [character](https://en.wikipedia.org/wiki/Token_(parser)); this will affect which stream is used for the redirection. The Unix standard I/O streams are:

|  |  |  |
| --- | --- | --- |
| **Handle** | **Name** | **Description** |
| 0 | [stdin](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_.28stdin.29) | Standard input |
| 1 | [stdout](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) | Standard output |
| 2 | [stderr](https://en.wikipedia.org/wiki/Standard_streams#Standard_error_.28stderr.29) | Standard error |

For example:

command1 2> file1

executes command1, directing the [standard error](https://en.wikipedia.org/wiki/Standard_streams#Standard_error_.28stderr.29) stream to file1.

In shells derived from [csh](https://en.wikipedia.org/wiki/C_shell) (the [C shell](https://en.wikipedia.org/wiki/C_shell)), the syntax instead appends the & (ampersand) character to the redirect characters, thus achieving a similar result. The reason for this is to distinguish between a file named '1' and stdout, i.e. 'cat file 2>1' vs 'cat file 2>&1'. In the first case, stderr is redirected to a file named '1' and in the second, stderr is redirected to stdout.

Another useful capability is to redirect one standard file handle to another. The most popular variation is to merge [standard error](https://en.wikipedia.org/wiki/Standard_streams#Standard_error_.28stderr.29) into [standard output](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) so error messages can be processed together with (or alternately to) the usual output. Example:

find / -name .profile > results 2>&1

will try to find all files named .profile. Executed without redirection, it will output hits to [stdout](https://en.wikipedia.org/wiki/Stdout) and errors (e.g. for lack of privilege to traverse protected directories) to [stderr](https://en.wikipedia.org/wiki/Stderr). If standard output is directed to file results, error messages appear on the console. To see both hits and error messages in file results, merge [stderr](https://en.wikipedia.org/wiki/Stderr) (handle 2) into [stdout](https://en.wikipedia.org/wiki/Stdout) (handle 1) using 2>&1 .

If the merged output is to be piped into another program, the file merge sequence 2>&1 must precede the pipe symbol, thus:

find / -name .profile 2>&1 | less

A simplified but non-POSIX conforming form of the command:

command > file 2>&1

is (not available in Bourne Shell prior to version 4, final release, or in the standard shell [Debian Almquist shell](https://en.wikipedia.org/wiki/Debian_Almquist_shell) used in Debian/Ubuntu):

command &>file

or:

command >&file

**NOTE:** It is possible to use 2>&1 before ">" but the result is commonly misunderstood. The rule is that any redirection sets the handle to the output stream independently. So "2>&1" sets handle 2 to whatever handle 1 points to, which at that point usually is *stdout*. Then ">" redirects handle 1 to something else, e.g. a file, but it does **not** change handle 2, which still points to *stdout*. In the following example. standard output is written to *file*, but errors are redirected from stderr to stdout, i.e. sent to the screen.

command 2>&1 > file

**Chained pipelines**

The redirection and piping tokens can be chained together to create complex commands. For example:

sort infile | uniq -c | sort -n > outfile

sorts the lines of infile in lexicographical order, writes unique lines prefixed by the number of occurrences, sorts the resultant output numerically, and places the final output in outfile. This type of construction is used very commonly in [shell scripts](https://en.wikipedia.org/wiki/Shell_script) and [batch files](https://en.wikipedia.org/wiki/Batch_file).

**Redirect to multiple outputs**

The standard command [tee](https://en.wikipedia.org/wiki/Tee_(command)) can redirect output from a command to several destinations.

ls -lrt | tee xyz

This directs the file list output to both standard output and the file xyz.

**See also**

* [Here-document](https://en.wikipedia.org/wiki/Here-document), a way of specifying text for input in command line shells
* [Shell shoveling](https://en.wikipedia.org/wiki/Shell_shoveling)
* [Command substitution](https://en.wikipedia.org/wiki/Command_substitution)
* [Process substitution](https://en.wikipedia.org/wiki/Process_substitution)

**External links**

* [dup](http://www.opengroup.org/onlinepubs/9699919799/functions/dup.html): duplicate an open file descriptor – System Interfaces Reference, [The Single UNIX Specification](https://en.wikipedia.org/wiki/Single_Unix_Specification), Issue 7 from [The Open Group](https://en.wikipedia.org/wiki/The_Open_Group)
* [Redirection Definition](http://www.linfo.org/redirection.html) by The Linux Information Project (LINFO)
* [I/O Redirection](http://tldp.org/LDP/abs/html/io-redirection.html) in [The Linux Documentation Project](http://tldp.org/)
* [Redirection in Windows](https://technet.microsoft.com/en-us/library/bb490982(en-us).aspx)
* [Creating a Child Process with Redirected Input and Output](http://msdn.microsoft.com/en-us/library/ms682499(VS.85).aspx) in Windows