



Speaking UNIX, Part 1: Command the power of the command line

Mix and match UNIX utilities to create impromptu programs

Level: Introductory

Martin Streicher (martin.streicher@gmail.com), Chief Technology Officer, McClatchy Interactive

07 Mar 2006

Learn the basics of the UNIX shell and discover how you can use the command line to combine the finite set of UNIX utilities into innumerable data transforms.

Speaking UNIX: Hello, shell

One of the most novel and differentiating features of a UNIX® system is its *command line*. With just a few keystrokes, including a bit of "glue", you can use the command line to combine the finite set of UNIX utilities into innumerable, impromptu data transforms.

For example, to find the list of unique filenames in the folder hierarchy rooted at the current working directory, you can type the following at your shell prompt:

```
find . -type f -print | sort | uniq
```

This command line combines three separate utilities:

- **find** plumbs the depths of the named directory -- in this case, the file system starting at `.` or *dot* (shorthand for the current working directory) -- and emits the names of all entries that match the given criteria. Here, `-type f` directs **find** to discover only plain files.
- **sort**, as its name implies, processes a list and emits a new list that's sorted alphabetically.
- **uniq** (pronounced "unique") scans a list, comparing adjacent elements in the list and removing any duplicates. For instance, suppose you have this list:

Listing 1. Example list

```
Groucho
Groucho
Chico
Chico
Groucho
Harpo
Zeppo
Zeppo
```

uniq reduces the list to the following:

Listing 2. uniq command

```
Groucho
Chico
Groucho
Harpo
Zeppo
```

However, if the original list of Marx Brothers is sorted first (reordering all occurrences of a name into a continuous run), running `uniq` yields this result:

Listing 3. Running `uniq`

```
Groucho      Chico
Groucho
Harpo
Zeppo
```

To learn more about the extensive features of `find`, `sort`, and `uniq`, refer to each utility's `man` page on your UNIX system.

Data in, data out, data all about

Used independently, `find` always takes the contents of the file system as its input data. However, both `sort` and `uniq` require data entry or input from *the standard input device* (stdin). Most often, you provide stdin using the keyboard: You type the data you want sorted on a series of lines, for example.

By default, `find` prints results on the *standard output device* (stdout), which is usually your terminal window. Both `sort` and `uniq` print outcomes to stdout.

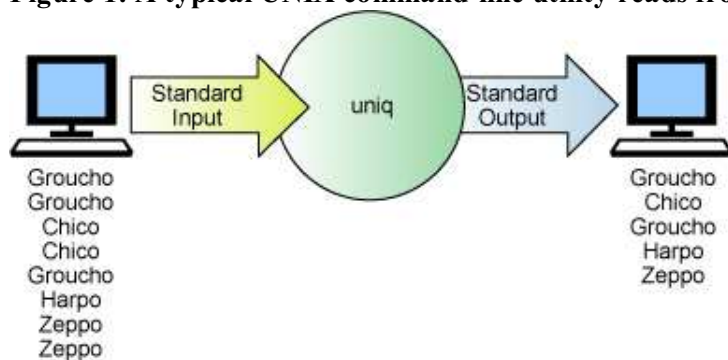
To demonstrate stdin and stdout, type the following text in your terminal window (assume that the leading percent sign (%) is your shell prompt):

Listing 4. stdin and stdout

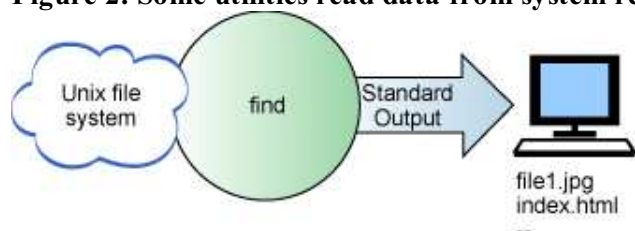
```
% sort
mustache
horn
hat
Control-D
```

`sort` reads the three lines you typed from stdin, sorts them, and writes the result to stdout. [Figure 1](#) presents a conceptual picture of running `sort`, and most UNIX command-line utilities, from the command line.

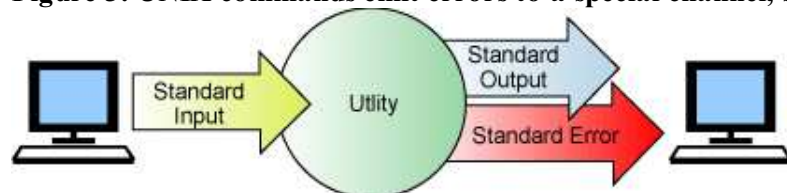
Figure 1. A typical UNIX command-line utility reads from stdin and writes to stdout



Some utilities, such as `find`, don't read from stdin. Instead, they read the data they should process from system resources, such as the file system or the system kernel, and write results to stdout. To visualize how `find` works, look at [Figure 2](#) below.

Figure 2. Some utilities read data from system resources and write results to stdout

In addition to using stdin and stdout, UNIX commands can emit error messages to a special outlet that's set aside (by convention, not mandate) for diagnostics. The outlet is called the *standard error device* (usually referred to as *stderr*). [Figure 3](#) illustrates a simple command line running a utility.

Figure 3. UNIX commands emit errors to a special channel, standard error

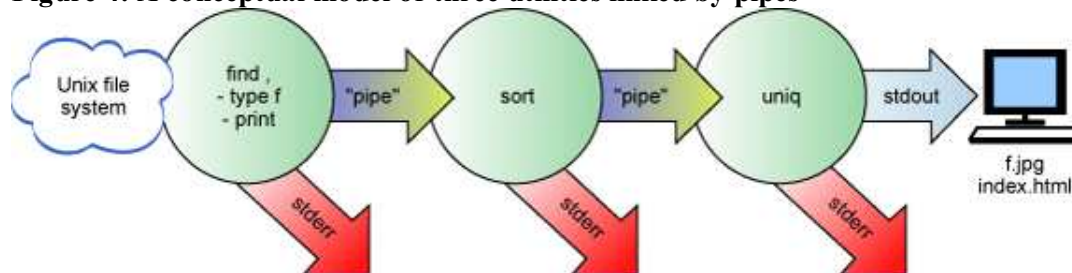
As shown in [Figure 3](#), most UNIX commands read input from the terminal, send results to the terminal, and print errors to the terminal. By default, and unless you specify otherwise, your terminal is the source of data for stdin and the destination for both stdout and stderr.

Routing traffic to and fro

However, you can change the source of stdin and the eventual destinations of both stdout and stderr. You can force stdin to read from a text file, a device (say, a probe connected to the computer), or a network connection. Comparably, you can send output to a file, a device, or a connection. In UNIX, where everything is a file, one source or destination is just as easy to consume or produce as another.

Changing the source and destination of a process's data is referred to as *redirection*. You redirect stdin to read data from a file or other source; and you can redirect stdout and stderr (separately) to write data somewhere other than the terminal window. In many cases, as in the original `find` command shown earlier, you can also redirect utilities to consume and produce data from and for other utilities. That is the purpose of the *pipe* (`|`). In a command, you can daisy-chain processes together using pipes, sending the data of one command to the next command, just like segments of copper pipe route water from your water heater to your sink.

[Figure 4](#) shows a conceptualization of the `find . -type f -print | sort | uniq` command.

Figure 4. A conceptual model of three utilities linked by pipes

The stdout of `find` becomes the stdin of `uniq`; in turn, the stdout of `uniq` becomes the stdin for `sort`. Finally, `sort` prints the results to its standard output device, which remains connected to the terminal. The stderr of the commands wasn't redirected, so all three utilities print error messages to the terminal. (Error messages from

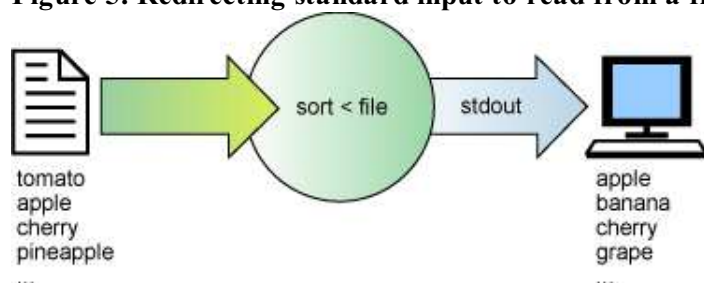
the three utilities are intermixed, but the order of the messages will be correct.)

If necessary, you can extend the pipeline shown above further and redirect the output of `uniq` to yet another utility. Just tack on another pipe to extend the transform further. For instance, you can append `| less` to paginate the output using `less`, or you can add `| wc -l` to find the number of unique filenames. (`wc` is an acronym for *word count*; `wc` can count characters, words, and lines.)

Alternatively, you can use `>` to save the output of the entire sequence to a file (destroying the existing contents of the file, if any). You can also use `>>` to *append* the results to an existing file (creating the file if it doesn't exist).

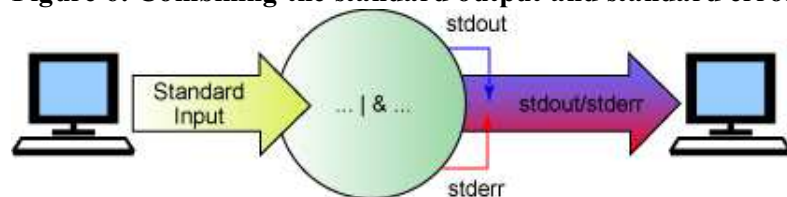
Another helpful redirection is `<`. [Figure 5](#) shows how `stdin` can be redirected to read from a file. The command `sort` reads a list of words from the named file and alphabetizes them.

Figure 5. Redirecting standard input to read from a file



Often, you'll want to capture `stdout` and `stderr`. For example, if you're running a large data-mining task, you might want to review the interim output and any errors that occurred during execution. You can use variants of the redirection syntax to do just that: `&`, `>&`, `>>&`, pipe, create, and append `stdout` and `stderr` simultaneously, respectively. [Figure 6](#) illustrates how `stdout` and `stderr` are combined into one output stream.

Figure 6. Combining the standard output and standard error devices



Introducing the Z shell

Most modern UNIX shells -- including the Bourne shell (`bash`) and the Korn shell (`ksh`) -- support the redirections mentioned here, although the specific syntax required by both of those shells might differ slightly. (Check your shell's documentation for specifics.).

Most of the redirection operators have been consistent features of all UNIX shells for at least 25 years. However, most of those shells have failed to break new ground and explore new ways to apply redirection. For instance, most shells can only redirect input from a single file, and you must use a utility like `tee` to output to more than one destination. (Like the tee junction used by plumbers, `tee` has one input and two outputs.) Here's an example using `bash` as the shell (the command-line interpreter):

Listing 5. bash example

```
bash$ ls
tellme
bash$ cat tellme
```

```

echo Your current login, working directory, and system are...
whoami
pwd
systemname
bash$ bash < tellme |& tee log
Your current login and working directory are...
strike
/home/strike
bash: systemname: command not found
bash$ ls
tellme log
bash$ cat log
Your current login and working directory are
strike
/home/strike
bash: systemname: command not found

```

Although UNIX shells are highly specialized and generally used interactively using the keyboard, a shell such as **bash** can also read input from a file. (After all, `stdin` is just a file.) In the previous snippet, the phrase `bash <` **commands** makes **bash** execute a list of commands found in the file `tellme`. The phrase `|&tee log` pipes the `stdout` and `stderr` of **bash** to the **tee** utility, which prints its `stdin` to `stdout` *and* to the file `log`.

But what if you want **bash** to process more than one input file? `cat file1 file2 file3 | bash` is a workable solution, and perhaps the only one, because **bash** doesn't support syntax like `bash < file1 < file2 < file3`.

Moreover, **bash** can't redirect output to more than one destination. For example, you can't enter an instruction like `bighairyscript > ~/log | mail -s "Important stuff" team` from the **bash** command line.

But a relatively new shell, the Z shell (**zsh**; see [Resources](#)), can process multiple input and output redirections within the same command line. For example, here's a command that saves `stdout` in a file called `log` and sends it to you using e-mail:

Listing 6. Z shell

```

zsh% bash < tellme > log | mail -s "Who you are" 'whoami'
bash: line 4: systemname: command not found
zsh% <log
Your current login, working directory, and system are...
strike
/home/strike

```

(The phrase `'whoami'` runs the command `whoami` and inserts the result of that command in place of the phrase. It's like running a little shell command before the rest of the command line runs.)

Let's walk through the previous command from left to right. The **bash** command creates the file `log` and mails the `stdout` of the commands found in `tellme` to you. Because `stderr` wasn't redirected by the `>` or the pipe, error messages are printed to `stdout`. The command `<log` is another Z shell shortcut; it's the same as `cat`. (And yes, the command `> file` is equivalent to `cat > file`.)

The Z shell can also process more than one input redirection. The Z shell command line `cat < file1 < file2 < file3` is the same as `cat file1 file2 file3`. Admittedly, the former syntax is more unwieldy than the latter and, in general, multiple `stdout` redirection is used far more often. However, if the utility you want to run doesn't accept multiple input arguments, the Z shell's multiple input redirection can come in handy.

The Z shell is full of other novel tricks, including better *globbing* (wildcard matches), advanced pattern matches, and an extensive automatic completion system that minimizes what you have to type at the command line. The next two articles in this series will delve further into the Z shell.

Shell tricks

Here are some powerful command-line combinations that are sure to make you more productive. The commands should work in all shells, not just `zsh`.

- Create a verbatim copy of any directory, including symbolic links, with `tar`:

```
tar cf - /path/to/original | \
(mkdir -p /path/to/copy; cd /path/to/copy; tar xvf -)
```

The first `tar` archives the directory `/path/to/original` and emits the archive file to stdout; the hyphen (`-`) used with the create (`C`) option specifies stdout. The command in parentheses is a subshell: Commands in the subshell don't affect the environment of the current shell. `mkdir -p` creates the named directory, including any intermediate directories that need to be created; and `cd` changes to the new directory. The second `tar` reads an archive from stdin and expands it in place; the hyphen used with the extract (`x`) option refers to stdin.

- To save the stdout of a command sequence and view it at the same time, use `less -0 file`. The `-0` option copies stdin to the named `file`. Here's an example:

```
sort /etc/aliases | less -0sorted
```

- If a directory contains thousands of files, your shell (including `zsh`, depending on the number of files and their names) might not be able to enumerate all the files using wildcard matches, because the command line is typically limited to a certain number of characters. Hence, shell script phrases such as

```
foreach i (*)
...
end
```

might fail. (You'll probably see a message like `Line length exceeded` when you exceed the length of your command line.) If such an error occurs, use a pipe and the `xargs` utility. `xargs` reads data from a pipe and runs a specified command for every line read.

For instance, if you want to find all Web pages on your server that reference `www.example.com`, you can use this command line:

```
% find / -name '*html' -print \
| xargs grep -l 'www.example.com' \
| less -0pages
```

`xargs` consumes the filenames from `find` and runs `grep -l` repeatedly to process every file, no matter how many files are named. (`grep -l` prints the name of the file if a match is found and then stops further matching in that file.) `less` allows you to page through the results and saves the list in the file named `pages`. The result is a list of filenames that contain the string `"www.example.com"`.

The journey begins

In this article, you learned the basics of the UNIX shell. Subsequent articles will delve more deeply into the plethora of command line tools and techniques at your disposal. From file systems to entire local area networks, virtually all information and systems administration can be undertaken efficiently from the UNIX command line.

Stay tuned!

Resources

Learn

- [Speaking UNIX](#) Check out other parts in this series.
- [zsh mailing list archive](#): Read this list to learn more Z shell tricks and tips.
- [AIX and UNIX](#): Want more? The developerWorks AIX and UNIX zone hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Get products and technologies

- [Z shell](#): Download the latest version of Z shell from the [Z shell home page](#).
- [IBM trial software](#): Build your next development project with software for download directly from developerWorks.

Discuss

- [zsh wiki](#): Collaborate, discuss, and share your zsh expertise.
- Participate in the [AIX and UNIX forums](#), [developerWorks blogs](#), and get involved in the developerWorks community.

About the author



Martin Streicher is the Chief Technology Officer of McClatchy Interactive and the Editor-in-Chief of [Linux Magazine](#). Martin holds a Masters of Science degree in computer science from Purdue University and has been programming UNIX-like systems since 1986. You can reach Martin at martin.streicher@gmail.com.

Share this....

 [Digg this story](#)  [del.icio.us](#)  [Slashdot it!](#)

UNIX is a registered trademark of The Open Group in the United States and other countries. Other company, product, or service names may be trademarks or service marks of others.