

---

# Speaking UNIX, Part 7: Command-line locution

## Expand your UNIX vocabulary

Skill Level: Intermediate

[Martin Streicher](mailto:martin.streicher@linux-mag.com) ([martin.streicher@linux-mag.com](mailto:martin.streicher@linux-mag.com))

Editor-in-Chief

Linux Magazine

06 Feb 2007

UNIX® has a dialect all its own, and its vocabulary of commands is quite large. But you don't have to learn everything all at once. Here, discover more command-line combinations and expand your mastery of the UNIX language.

Whenever you travel to a foreign country in which the inhabitants speak an unusual native tongue, you might arm yourself with essential survival phrases, such as "How much does this cost?", "What kind of meat is this?," and "Where is the bathroom?" Memorizing such little quips ensures that you don't get overcharged for that snake sandwich you ordered, and you know where to go when Mother Nature (or the snake sandwich) calls.

UNIX®, too, has a dialect all its own and, over the past six months, this *Speaking UNIX* series has provided something of a crash course in command-line locution. This month, learn several helpful phrases that will have you blending with the locals in no time. Grab your toothbrush, pack some comfortable shoes, and update your inoculations. You're off for sun, sand, and shells. (For the sun and sand, scoop up your laptop, head to the beach, plop down near the water's edge, and read this column. And don't forget your sunscreen.)

## Take the walking tour

Many of the previous *Speaking UNIX* columns (see [Resources](#)) have presented the `find` command -- an invaluable utility for scanning and processing files or even the entire UNIX file system. For instance, I often use `find` in combination with `grep` or Perl to process a large number of files. Need to know where a variable or constant is defined in a large body of code? Try this:

```
$ find /path/to/src -type f | xargs grep -H -I -i -n string
```

The output of the command is a list of file names that contain *string*, including the line number and the specific text that matched. The `-H` and `-n` options preface each match with the file name and line number of each match, respectively. The `-i` option ignores case. `-I` (capital "I") skips binary files.

If you haven't seen `xargs` before, it runs the command you specify -- here, `grep` with all the listed options -- once for each argument provided through standard input. Assuming that the `/path/to/src` directory contains files `a`, `b`, and `c`, using `find` in combination with `xargs` is the equivalent of:

```
grep -H -I -i -n string a
grep -H -I -i -n string b
grep -H -I -i -n string c
```

In fact, searching a collection of files is so common that `grep` has its own option to recurse a file system hierarchy. Use `-d recurse` or its synonyms `-R` or `-r`. For example, use:

```
$ grep -H -I -i -n -R string /path/to/src
```

This command achieves the same thing as `find/xargs`. (You'll find that many file-related UNIX utilities have options for recursion. `ls -R` recursively lists the contents of a hierarchy. `chmod`, `chgrp`, and `chown` use `-R` to apply mode, group, and ownership changes to an entire file system hierarchy. Be careful using `chmod -R`. You can inadvertently render a directory unusable if you remove its execute bits, say with `chmod -R a-x`. To be more selective, try `find . -type f | xargs chmod a-x`.)

So, when should you use `find/xargs`, and when should you use `grep`? When you need to be selective, use `find`. The `find` command has many conditionals that allow you to choose files that meet specific requirements, such as the conceptual "all regular files modified after midnight and owned by Joe." Otherwise, `grep -R` suffices nicely.

One other utility can be more convenient and *faster* to use than `find`. If you're trying to find a file by name, try `locate` instead of `find -name`. The `locate` command periodically (once a day or so, tuned by your system administrator) catalogs every file on your system and builds a database of path and file names. When you run `locate`, it scans its private database, trying to make matches.

For example, running the query `locate '*.1'` yields all files and directories whose names end with `.1`. (The leading asterisk means *match anything*.) For convenience, running the `locate fish` command is the same as `locate '*fish*'`.

## The currency exchange

Many UNIX utilities modify files. In most cases, altered contents are emitted to standard output where you can use redirection operators to further process (using pipes `-- |`) or capture the results (using the `>` or `>>` operators).

Other utilities, typically those that can process many files at a time, can retain the original file for safekeeping and generate a new file for the altered contents. For example, you can use Perl directly from the command line to process a file. The command:

```
$ perl -i.bak -pe 's/\bdollar(s?)/buck\l/g' file.txt
```

replaces "dollar" with "buck" and "dollars" with "bucks." The `perl -i` command changes `file.txt` in-place; `perl -i .bak` makes a copy of the original file and appends `.bak` to distinguish it from its new, modified version. Therefore, a command such as:

```
perl -i.bak -pe 's/\bdollar(s?)/buck\l/g' *
```

would create a backup for every file in the current directory. Assuming you had files `file1.txt`, `file2.txt`, and `file3.txt`, you would then have `file1.txt.bak`, `file2.txt.bak`, and `file3.txt.bak`. Mistakes happen, so making a backup is smart.

If you make an error and must recover an original, you might simply type:

```
mv file1.txt.bak file1.txt
```

But what happens if you have *hundreds of files* to rename? Certainly, you do not want to type hundreds of individual `mv` commands. Instead, you could type this:

```
foreach file in (*.txt)
do
    mv $file.bak $file
done
```

And that does work well in simple cases such as this one. However, this sort of task is so common that a special utility makes the job even faster. The command:

```
$ rename 's/\.bak$//' *.bak
```

performs the same task. The regular expression `s/\.bak$//` strips `.bak` from each file name listed on the command line -- here, `*`, or every file -- and uses the shortened name as the target file name.

The `rename` command works especially well when file names are particularly irregular. For instance, consider the contents of this directory, which looks something

like a collection of letters from a college freshman.

```
$ ls
RenT.txt  bEErMoNey.txt  gASmoNey.TXT
```

The `foreach` script shown above can't handle this problem, because the file names are so irregular. Yet `rename` can process it in a few keystrokes:

```
$ rename 'y/A-Z/a-z/' *
```

The `y` operator in the regular expression `y/A-Z/a-z/` is used for *translation*. Translation requires two lists: a list of original characters and a list of replacement characters. If the two lists are the same size, every instance of the first character of the original list in the text is replaced with the first letter of the replacement list. In other words, in the example, every instance of capital "A" is replaced with lowercase "a," "B" with "b," and so on. Lowercase letters in the text are left unchanged.

If you want to preview what `rename` does, add the `-n` option. This option shows the actions of the command without making the actual changes:

```
$ rename -n 'y/A-Z/a-z/' *
RenT.txt renamed as rent.txt
bEErMoNey.txt renamed as beermoney.txt
gASmoNey.TXT renamed as gasmoney.txt
$ rename 'y/A-Z/a-z/' *
$ ls
beermoney.txt  gasmoney.txt  rent.txt
```

One gotcha to avoid: On UNIX systems, file names are case sensitive. A directory can contain `Aa.Txt` and `aA.txT`. It's possible, as shown above, to write a `rename` rule that converts case-sensitive files into lowercase file names, thereby causing clashes among previously unique file names. What does `rename` do in that case? Let's see:

```
$ rename -n 'y/A-Z/a-z/' *
Aa.Txt renamed as aa.txt
aA.txT renamed as aa.txt
$ rename 'y/A-Z/a-z/' *
aA.txT not renamed: aa.txt already exists
$ ls
aA.txT  aa.txt
```

If you want to "clobber" existing files as the `rename` progresses, add the `-f` flag. In this example, the result would be one file named `aa.txt`. And which file was that originally? Because `rename` works in alphabetical order, the latter file `aA.txT` is now `aa.txt`. Why use `-f`? If two files are identical other than name, `rename -f` would remove duplicates.

## It's not clobberin' time

File management is certainly a large part of using UNIX. The system has innumerable configuration files. You likely have a plethora of data files and personal files. From time to time, you might accidentally remove or overwrite a valuable file. The shell and several file management utilities can help you avoid calamity.

Type the following commands at your shell prompt. These commands work in `bash`, but similar options exist in `zsh` and other shells.

```
$ alias mv=mv -i
$ alias rm=rm -i
$ set -o noclobber
```

The first two commands replace `mv` with `mv -i` and `rm` with `rm -i`, respectively, at the command line. Interactive mode forces you to confirm your action.

The third command provides a modicum of safety in the shell itself. With `noclobber` on, you can't accidentally overwrite a file using the `>` redirection operator:

```
$ ls
secret.txt
$ cat > secret.txt
bash: secret.txt: cannot overwrite existing file
```

To disable `noclobber`, type:

```
set +o noclobber
```

You can also force an overwrite at any time using the `>|` (a greater-than symbol followed by a vertical bar) redirection operator.

```
$ cat secret.txt
I love green eggs and ham.
$ echo "No more secrets" >| secret.txt
$ cat secret.txt
No more secrets
```

## Secrets of the locals

If you really want to discover a city, you have to visit the local watering holes. Here's a collection of command-line combinations worthy of Zagat.

`mkdir -p` makes quick work of hierarchy creation. With `-p`, `mkdir` creates each directory and subdirectory found in the named path:

```
$ mkdir -p make/many/directories/at/once
$ ls -R
./make:
many

./make/many:
```

```
directories
./make/many/directories:
at
./make/many/directories/at:
once
./make/many/directories/at/once:
```

If you ever need to know when your next pay day is, just type `cal`. With no arguments, `cal` prints the calendar for the current month. The `cal -3` command shows the previous, current, and next month, and `cal 06 2009` shows you June 2009. (My birthday is on a Monday that year!)

```
$ cal

    November 2006
Su Mo Tu We Th Fr Sa
    1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
$ cal 06 2009

    June 2009
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

Because UNIX has so many commands, it's virtually impossible to remember all the options for all the utilities. Indeed, sometimes you can't even remember the name of the utility.

When stumped, turn to `man`. Type `man man`, for example, to see how to use `man` itself. You can also see explanations for `rm` and `mv` using `man rm` and `man mv`. And, if you know what topic you're looking for, you can use `make -k` to find a list of `man` pages that relate to that topic.

```
$ man -k cron
cron (8)          - daemon to execute scheduled commands (Vixie Cron)
crontab (1)       - maintain crontab files for individual users (V3)
crontab (5)       - tables for driving cron
dh_installcron (1) - install cron scripts into etc/cron.*
```

Here, `man` discovered the `man` pages of those utilities whose one-line descriptions contained the word *cron*. Chances are, these `man` pages explain how to use `cron`, the system job-scheduling daemon.

And what are the little numbers for? Each number refers to a section of the online UNIX manual. Section 1 is reserved for all the commands a UNIX user can run in the shell. Section 5 describes file formats. Section 8 catalogs system administration commands. Other sections describe system calls (2), library calls (3), and more.

As you've seen, most commands emit output of one kind or another. Most command-line commands use standard output for results. But others use standard out and standard error to show progress and error messages, in that order. If you want to ignore that sort of output -- which is useful, because it often interferes with working at the command line -- redirect your output to the UNIX *bit bucket*, `/dev/null`. Bits check in, but they don't check out.

Here's a simple example:

```
$ ls
secret.txt
$ cat secret.txt
I am the Walrus.
$ cat secret.txt > /dev/null
$ cat socrates.txt > /dev/null
cat: socrates.txt: No such file or directory
$ cat socrates.txt >& /dev/null
$ echo Done.
Done.
```

If you redirect the standard output of `cat` to `/dev/null`, nothing is displayed, because all of the bits have been thrown into the virtual "permanent vertical file." However, if you make a mistake, error messages, which are emitted to standard error, are displayed. If you want to ignore all output, use the `>&` operator to send `stdout` and `stderr` to the bit bucket.

You can also use `/dev/null` as a zero-length file to empty existing files or create new, empty files:

```
$ cat secret.txt
Anakin Skywalker is Darth Vader.
$ cp /dev/null secret.txt
$ cat secret.txt
$ echo "The moon is made of cheese!" > secret.txt
$ cat secret.txt
The moon is made of cheese!
$ cat /dev/null > secret.txt
$ cat secret.txt
$ cp /dev/null newsecret.txt
$ cat newsecret.txt
$ echo Done.
Done.
```

By the way, if you happen to use UNIX on the Macintosh, try the `open` command in a Terminal window. For instance, if the current working directory has a file named `poodle.jpg`, the command `open poodle.jpg` launches Preview, the built-in Mac OS X image viewer, with the picture of your poodle. Mac OS X `open` is a great link between the command line and the windowing environment of the Macintosh -- and it is often far faster than resorting to the Finder.

## Have to visit again!

Whew! That was a fast-paced class, but now you're prepared to travel farther into UNIX. You even know where the bit bucket is when Mother Nature calls.

As always, there's more to cover. In coming months, the *Speaking UNIX* series delves into job control, regular expressions (a strange dialect, to be sure, but not impossible to master), how to build new utilities you download from the Internet, and more.

Don't forget your sunscreen.

**Share this...**

[Digg  
this  
story](#)

[Post  
to  
del.icio.us](#)

[Slashdot  
it!](#)



# Resources

## Learn

- [Speaking UNIX](#): Check out other parts in this series.
- [RSS](#) feed for this series. (Find out more about [RSS](#).)
- Check out other articles and tutorials written by Martin Streicher:
  - [AIX® and UNIX zone](#)
  - [Across developerWorks](#)
- Search the AIX and UNIX library by topic:
  - [System administration](#)
  - [Application development](#)
  - [Performance](#)
  - [Porting](#)
  - [Security](#)
  - [Tips](#)
  - [Tools and utilities](#)
  - [Java™ technology](#)
  - [Linux®](#)
  - [Open source](#)
- [AIX and UNIX](#): The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- [New to AIX and UNIX](#): Visit the New to AIX and UNIX page to learn more about AIX and UNIX.
- [AIX 5L™ Wiki](#): A collaborative environment for technical information related to AIX.
- [Safari bookstore](#): Visit this e-reference library to find specific technical resources.
- [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

- [IBM trial software](#): Build your next development project with software for

download directly from developerWorks.

## Discuss

- Participate in the [developerWorks blogs](#) and get involved in the developerWorks community.
- Participate in the AIX and UNIX forums:
  - [AIX 5L -- technical forum](#)
  - [AIX for Developers Forum](#)
  - [Cluster Systems Management](#)
  - [IBM Support Assistant](#)
  - [Performance Tools -- technical](#)
  - [Virtualization -- technical](#)
  - [More AIX and UNIX forums](#)

## About the author

Martin Streicher

Martin Streicher is a Web applications developer at [hesketh.com](#) in Raleigh, North Carolina, and he is the Editor-in-Chief of [Linux Magazine](#). Martin holds a master's degree in computer science from Purdue University and has been programming UNIX-like systems since 1986. You can reach Martin at [martin@hesketh.com](mailto:martin@hesketh.com) or [martin.streicher@linux-mag.com](mailto:martin.streicher@linux-mag.com).

## Trademarks

IBM, AIX, and AIX 5L are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.