Open Computing ‘‘Hands-On’’ Tutorial: January 1994

Explosion in a Punctuation Factory

The rewriting rules of Sendmail help your system check and correct an electronic mail address before sending it to its final destination

By Bryan Costales

The Sendmail program is the mail-transfer software for many Unix systems, but Sendmail’s configuration file has a long and glorious history of being difficult to understand, much less modify. Are Sendmail’s rewriting rules confusing to you? If they are, you’re not alone. The rewriting rules--used to rewrite mail headers, check for errors, and to select mail programs--don’t have to be all that mysterious. Compact, yes, but relatively simple once you begin to understand them.

The rewriting rules have been variously described as resembling: modem noise, Mr. Dithers swearing in the comic strip ‘‘Blondie,’’ and an explosion in a punctuation factory. While these allusions are sadly true, they are also, in reality, misleading. What appears confusing and complex is, in reality, just succinct.

The Sendmail program parses (reads and processes) each rule every time it reads its configuration file, `sendmail.cf`. Because that process needs to be swift, rules have been designed to be easier for Sendmail to parse than for you to understand.

Why Rules?

The rules are used to modify mail addresses, to detect errors in addressing, and to select an appropriate means of mail delivery. Addresses need to be modified because they can be specified in many ways yet are required to be in specific forms for particular means of delivery.

To illustrate, consider the address `friend@uuhost`. If the machine named ‘‘uuhost’’ were connected to yours over a dial-up line, the message would likely be sent using UUCP software. That software requires addresses to be expressed in UUCP form `uuhost!friend`. The Sendmail rewriting rules control the transformation.

Another role for the rules is to detect (and reject) errors locally. This filtering prevents errors from propagating over the network. Mail to an address without a user name, such as `@neighbor`, is one such error. It is better to detect this kind of error locally rather than having the host ‘‘neighbor’’ reject it.

Sequences of rules are grouped together into rule sets. Each set is similar to a subroutine. A rule set is declared with the ‘‘S’’ key letter, which must begin a line in the Sendmail configuration file. For example, ‘‘S0’’ begins the declaration of the rules that forms rule set number 0. Rule sets are numbered starting from 0, where sets 0 through 5 are internally defined by Sendmail to have very specific purposes:

```
0 Resolve delivery agent
1 Process sender address
2 Process recipient address
3 Preprocess all addresses
4 Postprocess all addresses
5 Rewrite unaliased
```

Rule set definitions may appear in any order in the configuration file. For example, rule set S5 may be defined first, followed by S2 and then S7. The rule sets are gathered when the configuration file is read, and they are sorted internally by Sendmail.

If a rule set is undefined, the result is the same as if it were defined but had no rules associated with it. It is like a subroutine that contains nothing but a ‘‘return’’ statement. It does nothing and produces no errors.

To observe the effect of rules that do nothing, create a three-line configuration file named, say, `x.cf` [as shown in Listing 1A] and run Sendmail in rule-testing mode on that file with the command shown [in Listing 1B].

The `-bt` command-line switch causes Sendmail to run in address-testing mode. In this mode, Sendmail waits for you to type a rule set and an address. It then shows you how the rule set ‘‘rewrites’’ the address. As Listing 1B shows, you enter an address by specifying a rule set number and then a space and a mail address. The rule set specified is 0, but you can specify any number.

The ‘‘rewrite:’’ designation that begins each line of address- testing-mode output is simply there to highlight rewriting lines when they are mixed with other kinds of debugging output. The ‘‘input’’ designation means that Sendmail placed the address into the workspace (more about this later). The ‘‘returns’’ designation shows the result after the rule set has rewritten that address based on its rules.

The address that was fed to Sendmail (bob@here) was first split into parts (tokens) based on the separating characters defined by the ‘‘Do’’ macro shown in Listing 1A, and 10 others defined internally by Sendmail, namely: `( ) < > , ; \ " \r \n`

For clarity, each token in Listing 1B was printed within full quotation marks; however, some versions of Sendmail omit these marks. The ‘‘input:’’ line shows the three tokens passed to rule set 0. The ‘‘returns:’’ line shows, because there is no rule set 0, that the undefined (empty) rule set returns those tokens that make up the address unmatched and unchanged.

The example illustrates version 8 Sendmail. If you are running an old version of Sendmail, two things will be different. First, the initial output will not include the message ‘‘(ruleset 3 NOT automatically invoked)’’, but will include two extra rewrite lines. Second, old versions of Sendmail always assume you want to see the effect of rule set S3, whether you do or not.

Rule Sets

Each rule set may contain any number of individual rules or none at all. Rules begin with the ‘‘R’’ key letter and generally take the following general form:

```
S0
```

```
Rlhs rhs

Rlhs rhs comment
```

The first line--the S0--declares the start of rule set 0. All the lines after the S line that begin with R belong to that rule set. A new rule set begins when another S line with a different number appears.

Each R line is an individual rule in a series of rules that form a rule set. If you examine the Sendmail configuration file for almost any major mail-handling site you'll see that any given rule set can have a huge number of rules. But our hypothetical rule set 0 has only two rules and therefore only two lines that begin with an R.

Each rule has two distinct parts, each divided from the other by one or more tab characters. You can use space characters inside each part, but you must use tabs to separate the parts.

The left-hand part of the rule is called the *lhs* for left-hand side. Conversely, the right-hand part is denoted *rhs*. These two form the rule. A comment may optionally follow the right-hand side, and, if present, must be separated from it by one or more tab characters.

The left-hand and right-hand sides form a ''do while'' pair. As long as the left-hand side evaluates to true, the right-hand is processed. If the left-hand side evaluates false, Sendmail skips to the next rule for that rule set.

The Workspace

Whether the left-hand side is true or false is determined by making comparisons. When an address is processed for rewriting by a rule set, Sendmail first separates the parts into tokens and stores those tokens internally in a buffer called the ''workspace.''

When the left-hand side of a rule is evaluated, it is divided into tokens and those are compared to the tokens in the workspace. If both the workspace and the left-hand side contain exactly the same tokens, a match is found, and the result of the left-hand side comparison is true. To illustrate, in Listing 2A we've added two lines to the end of our minimal configuration file, `x.cf`. Don't forget that the three parts of the rule are separated from each other by tab characters. This example creates a ''demo'' rule set that illustrates a few introductory concepts about rules.

Now run Sendmail in rule-testing mode, as shown in Listing 2B. As we did in Listing 1B, enter rule set 0 and a typical e-mail address at the prompt. Notice that nothing was rewritten, even though there is a rule set 0 and a rule in our sample configuration file. Remember that a rule is only rewritten if the workspace and the left-hand side exactly match. For the demo rule, they do not match (see Figure 1).

Enter the exact text that appears in the left-hand side of the demo rule at the prompt (see Listing 2C). An amazing thing happens. The rule has actually rewritten an address. The address ''left.side'' was given to rule set 0 and was rewritten by the rule in that rule set to become the address ''new.stuff''. This transformation was possible because the workspace and the left-hand side exactly matched each other, so the result of the left-hand side comparison was true.

Before leaving this demo rule set, perform one final experiment. Enter the text ''left.side'' again, but this time change the case of the letters to upper case. Notice that the workspace and the left-hand side

still match, even though they now differ by case. This example illustrates that all comparisons between the workspace and the left-hand side of rules are done in a case-insensitive manner. This property enables rules that solve complex problems to be written without the need to distinguish between upper- and lower-case letters.

The Flow of Addresses Through Rules

When rule sets contain many rules, the ‘‘flow’’ is from the first through the last rule (top to bottom), in the order they are declared in the configuration file.

To illustrate, modify the two demo lines you added to the sample configuration file, replacing them with the three new demo rules shown in Listing 3. There are only two parts to each rule (the comment is missing).

Before you test these new rules, consider what they do. The first rule rewrites any ‘‘x’’ in the workspace into a ‘‘y’’. The second rule rewrites any ‘‘y’’ in the workspace into a ‘‘z’’. And the last rule rewrites any ‘‘z’’ that it finds in the workspace into an ‘‘a’’.

Now run Sendmail in rule-testing mode once again, and, one at a time, enter rule set 0 and one of the letters ‘‘x’’, ‘‘y’’, and ‘‘z’’. No matter which of ‘‘x’’, ‘‘y’’, or ‘‘z’’ you enter, each is rewritten into ‘‘a’’, illustrating the ‘‘flow’’ of addresses (the workspace) through rules.

Let’s look in detail at what is going on by examining the input. Follow along with Figure 2. When you first enter rule set 0, the first rule of that rule set tries to match its left-hand side to the workspace; the left-hand side exactly matches the workspace, so the right-hand side rewrites the workspace so that ‘‘x’’ is replaced by ‘‘y’’.

Now the next rule tries to match its left-hand side to the workspace. But what is contained in the workspace has been rewritten by the first rule. The key point here is that each rule compares its left-hand side to the current contents of the workspace, even though they may have been rewritten by earlier rules. It should now be clear why all three letters are rewritten to ‘‘a’’ (see Figure 3). Now feed one more letter into Sendmail in rule-testing mode. This time enter anything other than an ‘‘x’’, ‘‘y’’, or ‘‘z’’, say the letter ‘‘b’’. Notice that the workspace remains unchanged because ‘‘b’’ did not match of the left-hand sides in any of the three rules. If the left-hand side of a rule fails to match the workspace, that rule is skipped, and the workspace remains unchanged.

Operators Versus the Workspace

Rules would be pretty useless if they always had to match the workspace exactly. Fortunately, that is not the case; in addition to literal text, you can also use operators. Operators are like wild cards in that they allow the left-hand side of rules to match arbitrary text in the workspace. To illustrate, look at Figure 4. The left-hand side begins with the first character following the ‘‘R’’ key letter. The left-hand side in Figure 4 is the operator, `$+`, the truth of which is determined by a process called pattern matching. The left-hand side `$+` (a single operator) is a pattern that means ‘‘match one or more tokens.’’

The address being evaluated is separated into tokens, placed into the workspace (see Figure 5), and then the workspace is compared to that pattern. When matching the workspace to a left-hand side pattern, Sendmail scans the workspace from left to right. Each token in the workspace is compared to the

operator (`$+`) in the left-hand side pattern. If the tokens all match the pattern, the left-hand side is true.

The `$+` operator simply matches any one or more tokens. As you can see, if there are *any* tokens in the address at all (the workspace is not empty), the left-hand side rule `$+` evaluates to true.

A rule using `$+` on the left-hand side is not sufficient to handle all possible addresses, especially bad addresses (see Figure 6). To make matching in the left-hand side more effective, Sendmail allows literal text to appear in the pattern. To make sure that the address in the workspace contains a user part and a host part separated by the `@` character, the left-hand side pattern `$+@$+` can be used. Just like the address in the workspace, this pattern is separated into tokens before it is compared for a match. Operators (like `$+`) are handled individually, and the `@` is a token because it is a separator character defined by the ‘‘Do’’ macro definition.

The `$+@$+` pattern is separated into three tokens: `$+`, `@`, and `$+`. Text in the pattern must match text in the workspace exactly, token for token, if there is to be a match. A good address in the workspace--one containing a user part and a host part--will match our new left-hand side (`$+@$+`) as shown in Figure 7. The ‘‘flow’’ of matching begins with the first `$+`, which matches one token of the one or more tokens in the workspace. The `@` matches the identical token in the workspace. At this point, the `$+@` part of the pattern has been satisfied. All that remains is for the final `$+` to match one or more of all the remaining tokens in the workspace.

But a bad address in the workspace will not match. For example, consider an address that lacks a user name (as shown in Figure 8). The first `$+` incorrectly matches the `@` in the workspace. Because there is no other `@` in the workspace to be matched by the `@` in the pattern, the first `$+` matches the entire workspace. Because there is nothing left in the workspace, the attempt to match the `@` fails.

When any part of a pattern fails to match the workspace, the entire left-hand side fails. One small bit of confusion may yet remain. When an operator like `$+` is used to match the workspace, Sendmail always does a minimal match. That is, it only matches what it needs to for the next part of the rule to work. Consider a left-hand side of `R$+@$+`, in which the first `$+` matches everything in the workspace up to the first `@` character in the workspace. For example, for a workspace of `a@b@c`, the `$+@` causes the `$+` to match only the characters--the a--up to the first `@` character. This character is the minimum that needs to be matched, and so it is the maximum that will be matched.

More Play With Left-Hand Side Matching

Take a moment to revise the sample Sendmail configuration file as shown (Listing 4). I’ve given each temporary right-hand side a number to see whether it is selected. The `$@` in front of each right-hand side prevents any successful rewrite being carried to any subsequent rules. Now run Sendmail in rule-testing mode again. The first address to specify is an ‘‘@’’ which returns one. The ‘‘@’’ causes the first right-hand side to be selected. The left-hand side--the pattern to match--contains the lone `@`. That pattern matched the tokenized workspace `@` exactly, so the right-hand side for that rule is returned. No other rules are called because of the `$@` prefix.

Next enter an address that contains just a host and domain part, but not a user part, something like ‘‘@host.domain’’. The first thing to notice is what was not printed! The workspace does not match the pattern of the first rule. But instead of returning an error, the workspace is carried down as is to the next rule, where it does match.

Now enter an address that fails to match the first two rules but successfully matches the third, something like ‘‘user@host.domain’’. The flow for this address is shown in Figure 9.

The fourth rule contains the original lone `$+`, which is there to catch any addresses that slip past the first three. Go ahead and test it. Try addresses like your log-in name or UUCP addresses like ‘‘user@host.uucp’’ and ‘‘host!user’’. Can you predict what will happen with weird addresses like ‘‘@@’’ or ‘‘a@b@c’’?

Other Operators

A single operator, the `$+`, allowed a rule set to be designed with four rules. Far more complex rule sets become possible when you take advantage of Sendmail's other left-hand side operators. Here's a list:

```
$@ Exactly none

$* Zero or more

$+ One or more

$- Exactly one
```

But the story doesn't end here. In this article you've been given a glimpse of how Sendmail's rules work. In all the listings, I've shown only ordinary, literal text in the right- hand side. The power of Sendmail lies in its use of operators in the right-hand side to rewrite addresses in complex and sophisticated ways. The right-hand side operators are:

```
$: Rewrite once (prefix)

$@ Return (prefix)

$digit Copy by position

$( Database lookup

$[ Name canonicalization
```

Clearly, there is not enough room in this tutorial to go over all the possible Sendmail rewriting rules. And the rewriting rules are only a part of Sendmail. The Sendmail program is a very flexible tool, and its configuration file reflects this flexibility by its complexity. Still, this tutorial hopefully has shown that you can understand Sendmail's configuration file, and encouraged you to continue exploring.

Copyright © 1995 The McGraw-Hill Companies, Inc. All Rights Reserved.
Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Go To Content

Search

Last Modified: Thursday, 30-Nov-95 16:53:27 PST