

CGI Forms on RCI

Platform: UNIX

Level of Difficulty: Advanced

This program is designed to be used as a CGI program to process input from forms. It is intended only for use with the POST method. Here is typical HTML:

```
<FORM ACTION="/cgi-bin/form/~smith/myform" METHOD=POST>  
Var 1: <INPUT TYPE="text" name="var1"><br>  
Var 2: <INPUT TYPE="text" name="var2"><br>  
<INPUT TYPE="submit">
```

The action must specify the URL where the form program is installed, in this case /cgi-bin/form, followed by the location of a set of files that describe how the form is to be processed. In this case the files are specified by ~smith/myform.

The program actually uses a set of files. For example when the argument is ~smith/myform, specifications will be read from ~smith/html_data/myform.form-spec, and the data will be put in the file ~smith/html_data/myform.form-data. Additional suffixes may be used as described below. In the action you may also specify ~smith/myform.form-spec. The .form-spec will be removed before producing other filenames such as myform.form-data.

NOTE: ~smith means ~smith/html_data. Just as ~smith in a normal web page refers to a subdirectory ~smith/public_html, ~smith in a form refers to ~smith/html_data. A separate directory is used for security reasons.

Data from the form can be put in a file and/or mailed. The processing of the data is controlled by the form-spec file. The form-spec file contains lines, each of which are of the form variable=value. E.g.

```
sizelimit=20  
replace=*/"/"/"  
test=*/  
fileline="<$var1>","<$var2>"  
mailline="<$var1>","<$var2>"  
mailto=smith@rci
```

In general the lines are executed in the order they occur in the file. The order shown above is a typical one. With this form-spec file, the data would be placed into the form-data file and also sent via email. That is because both fileline= and mailto= commands appear in the file.

WARNING: the keywords must be given exactly as shown. Matches are case-sensitive. Blanks are significant. Unrecognized lines are ignored, so you can put comments in. (In general any line without an = sign should be safe, even if I add new variables.)

"fileline=" indicates that data from the form is to be written to a file. It will write to the filename given in ACTION, with .form-data added to the end. Everything after the = will be written to the file, except that <\$xxx> will be replaced by the value of the variable xxx from the form. Form writes the data at the time it processes the fileline= command. (<\$xxx> can also be used to refer to environment variables, e.g. <\$\$REMOTE_USER>. These environment variables are set by the web server, as defined in the CGI specification. Since the exact set of variables depends upon the specific server and configuration, they are not described here.) If you put in <\$foo>, and there is no variable foo in the form, a null string will be used. The final line may not be longer than 8KB.

"mailline=" indicates that the data from the form is to be sent via email. It is handled exactly like "fileline=". That is, <\$xxx> is replaced with the value of xxx. The mail is not actually sent until the "mailto=" line is processed.

"mailto=" indicates that the data from the form is to be sent via email. It will be sent to the address you list after the =. The mail is sent at the time form processes the "mailto=" line. There are two ways to specify what goes in the mail message. Either put a "mailline=" command before the "mailto=" command, or create a mail template file. A mail template file has the usual file name, but ends in .form-mail. The template file should contain the entire email message, complete with all headers. Anywhere <\$xxx> appears in the template file, it is replaced with the value of the form variable xxx. The mailto= may list more than one address, separated by spaces. Variable substitution is done in the mailto= line. The final line may not be longer than 8KB.

"test=" gives a pattern that is used to test one or more variables. Often you want to make sure the user filled in certain fields, or that certain fields are numeric, etc. test= allows you to test the data the user filled in. If the test fails, an error message will be shown. No data will be put in the file and no data will be sent. The format of a test= line is

```
test=variable/regexp/error message/
```

For example,

```
test=name/ [a-zA-Z] [a-zA-Z] */The name must be alphabetic/
```

will test the variable "name". It must match the pattern [a-zA-Z] [a-zA-Z] *, which only allows letters. If the test fails, the user will get the error message "The name must be alphabetic". You can use * in place of the variable name. If you do, that test will be applied to all variables. You can omit the error message. If you do, form will generate a message. You may omit the test, or leave it blank, i.e. //. If you do, form will make sure that the user specified something for the field, but will not do any additional checking. The simplest test is the one I used as the example: test=*//. This checks all variables, and requires that they were all filled in. For details on the patterns, use the command "man regexp". Not that the delimiter can be /, \, ", or ` . It is possible to test an environment variable by putting a \$ before the variable name, e.g. "test=\$REMOTE_USER//" will require that the environment variable REMOTE_USER has been set. However, test=* tests only the variables from the form, not the environment variables.

NOTE: test=* tests all variables that are passed by the form. This may not be what you want. As long as you know that the form will always send the required variables, test=* is fine. For normal text boxes, the form will always send something, even if it is a null string. So if your form consists entirely of text fields, test=*// is enough to guarantee that all boxes are filled in. However, checkboxes only generate data if they are filled in. For cases like this, you may need to put in explicit tests for all the variable that you regard as required.

"replace=" allows you to change some letters into other letters. Normally you are going to be producing files that use commas as separators, and possibly " to quote text. If the user puts a comma or " in the data, this could confuse your database program. The replace command lets you replace , or " with something else. If you are using a comma separated list, you might do something like

```
replace=* / , / $ /
```

That would replace all commas with dollar signs in all variables. replace= has the general syntax

```
replace=variable/regexp/text/
```

As with test, you can apply the replacement to the value of an individual variable, or you can use * to do it for all variables. The delimiter can be any of /, \, ", or ` . You cannot do replacements on environment variables.

"sizelimit=" allows you to put a limit on the size of the data file that is written by the "fileline=" command. The limit is in bytes. This prevents users from feeding lots of junk to your form and filling up your disk quota. Note that form will accept a maximum of 8K of data from a single form. The "sizelimit=" must appear before the "fileline=".

As described above, form will use several files, all of which are in ~user/html-data. The filenames are based on a path specified in the ACTION attribute in your <FORM> tag. (It is also possible to specify a path using a variable FORM_DATAFILE in the form itself. In theory you could set up a form that can be processed by different form specifier files, depending upon which box a user checks. This is almost certainly too much rope.) All files must be owned by the same user. They must not be symbolic links.

If you want to write data to a file, the form program must be able to write to the file. Normally this is done by setting the form-data file so that the group "cgi" can write to it. The program "make_html_data" will setup an html_data subdirectory for you in such a way that all files in it are automatically in group cgi. You must still make sure that the form-data file is group writable.

Here is a list of the different suffixes used:

form-spec: This file is required. It specifies how the form is to be processed, using the variables described above.

form-data: This file is required if you want to write data to a file. The file must exist, and must be writable by the form program. If you just want to mail your data, you don't need it.

form-mail: This file contains a template for email. It is only needed if you want to send mail. If you are sending email, you must either specify `mailline=` in the form-spec file, or supply the form-mail file. It does not make sense to do both.

form-error: This file contains a template for error messages. It should be a web page, with all the necessary tags. The pseudo-tag `<ERROR>` will be replaced by the text of the error message. Only the first occurrence will be replaced. This file is optional. If you don't have it, form will use a default error message format.

form-ok: This file contains a template for the success page. Once form has accepted the user's form, it will show this page. It should be a web page, with all of the necessary tags. Pseudo-tags that look like `<$xxx>` will be replaced with the value of the form variable xxx. This file is optional. If you don't have it, form will use a default success page format.

RECOMENDED SETUPS

I expect the most common use for this program will be to prepare data files which you will read into a spreadsheet or database program. Most commonly, these programs expect fields separated by commas or tabs. If you have a form with variables called "name" and "address", here is what you need in the form-spec file to generate a comma-separated file:

```
fileline=<$name>,<$address>
```

Some programs want you to put quotes around the variables, in case there are blanks or commas in them. Here is what you would do to put quotes around those two variables:

```
fileline="<$name>",<$address>"
```

Finally, I recommend that you specify replacements for the delimiters. If you are using a simple comma separated list, you'll need to do a replacement for the comma at the end of the line. Otherwise a comma in the name or address could confuse the database program. In the following example, I replace comma and newline with dollar. (`\n` is the special code indicating the newline character)

```
replace=*, / $ /
replace=*, \n / $ /
fileline="<$name>",<$address>"
```

If you are using quotes around everything, you don't need to worry about the commas. But you do have to worry that a user may put quotes in their data. The following example replaces quotes with single quotes, and turns new lines into dollars:

```
replace=*, / ' /
replace=*, \n / $ /
fileline="<$name>",<$address>"
```

Once you have gotten the fileline and replace commands right, think about whether you want to do any tests. The easiest tests are to see whether the user typed any data. If all of the fields are required, you want to make sure the user entered something for every field. The following will do that:

```
test=*/ /
replace=*, / ' /
replace=*, \n / $ /
fileline="<$name>",<$address>"
```

If only certain fields are required, you will want to use a separate test command for each. For example if only name and address are required, you might have the following:

```
test=name // You must fill in your name/  
test=address // You must fill in your address/  
replace=* /, /' /  
replace=* / \ n / $ /  
fileline="<$name>","<$address>","<phone>"
```

You probably do not want to get into more complex tests unless you are an Unix expert.

SAMPLES

To create a form that sends the results to a *data file* only.

Create an HTML file

```
<BODY>  
<FORM ACTION="/cgi-bin/form/~smith/myform" METHOD=POST>  
<P>Name  
<INPUT TYPE="text" name="name"><BR>  
<P>Address  
<INPUT TYPE="text" name="address"><BR>  
<P>  
<INPUT TYPE="submit">  
</BODY>
```

Replace ~smith with your ~username. Save the file as "form.html" and put it in your public_html directory.

Using a telnet program login to your RCI account.

```
make_html_data  
cd html_data  
pico myform.form-spec
```

Place this text in the myform.form-spec file

```
sizelimit=200  
replace=* /, /' /  
replace=* / \ n / $ /  
test=*//  
fileline="<$name>","<$address>"
```

Type Ctrl-X to save the file.

```
touch myform.form-data
```

Fill out the form on the World Wide Web to test it. Look in the file /html_data/myform.form-data, your results should be there.

To create a form that sends the results to a *data file and email*.

Create an HTML file

```
<BODY>
<FORM ACTION="/cgi-bin/form/~smith/myform" METHOD=POST>
<P>Name
<INPUT TYPE="text" name="name"><BR>
<P>Address
<INPUT TYPE="text" name="address"><BR>
<P>
<INPUT TYPE="submit">
</BODY>
```

Replace ~smith with your ~username. Save the file as "form.html" and put it in your public_html directory.

Using a telnet program login to your RCI account.

```
make_html_data
cd html_data
pico myform.form-spec
```

Place this text in the myform.form-spec file

```
sizelimit=200
replace=*,/' /
replace=*\ n / $ /
test=*/
fileline="<$name>","<$address>"
mailline="<$name>","<$address>"
mailto=smith@rci
```

Replace "smith" with your "username". Type Ctrl-X to save the file.

```
pico myform.form-mail
```

Place this text in the myform.form-mail file

```
From: "My form page"
To: smith@rci.rutgers.edu
Subject: Personal Information from myform
```

```
Personal Information:
Name: <$name>
Address: <$address>
```

Replace "smith" with your "username". Type Ctrl-X to save the file.

```
touch myform.form-data
```

Fill out the form on the World Wide Web to test it. Look in the file /html_data/myform.form-data, your results should be there. Also, check your mail to see if the formatted results were sent to you.

To create a form that sends the results to a *data file and formatted email*.

Create an HTML file

```
<BODY>
<FORM ACTION="/cgi-bin/form/~smith/myform" METHOD=POST>
<P>Name
<INPUT TYPE="text" name="name"><BR>
<P>Address
<INPUT TYPE="text" name="address"><BR>
<P>
<INPUT TYPE="submit">
</BODY>
```

Replace ~smith with your ~username. Save the file as "form.html" and put it in your public_html directory.

Using a telnet program login to your RCI account.

```
make_html_data
cd html_data
pico myform.form-spec
```

Place this text in the myform.form-spec file

```
sizelimit=200
replace=*,/' /
replace=*\ n / $ /
test=*/
fileline="<$name>","<$address>"
mailline="<$name>","<$address>"
mailto=smith@rci
```

Replace "smith" with your "username". Type Ctrl-X to save the file.

```
pico myform.form-mail
```

Place this text in the myform.form-mail file

```
From: "My form page"
To: smith@rci.rutgers.edu
Subject: Personal Information from myform
```

```
Personal Information:
Name: <$name>
Address: <$address>
```

Replace "smith" with your "username". Type Ctrl-X to save the file.

```
touch myform.form-data
```

Fill out the form on the World Wide Web to test it. Look in the file /html_data/myform.form-data, your results should be there. Also, check your mail to see if the formatted results were sent to you.