Expert Shell Scripting

Copyright © 2009 by Ron Peters

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-1841-8

ISBN-13 (electronic): 978-1-4302-1842-5

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Frank Pohlmann
Technical Reviewer: Brian Culp
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Sofia Marchant
Copy Editor: Candace English
Associate Production Director: Kari Brooks-Copony
Production Editor: Liz Berry
Compositor: Pat Christenson
Proofreader: Lisa Hamilton
Indexer: Julie Grady
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at http://www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

www.itbook.store/books/9781430218418

CHAPTER 23

Command-Line E-mail Attachments

I often send e-mail to myself containing information gathered from a running system. Most of the information comes as flat text obtained from various files, or output from system commands. From time to time the file that I would like to send is a binary of some type. Sometimes I want a file to show up as an attachment regardless of whether it is a binary.

I wrote a few scripts that can perform such a task. They encode the binary file as flat text for transmission and then e-mail the file. The file can then be decoded at the receiving end manually or, more conveniently, by the e-mail client receiving the file. Each script takes as input the binary file and the destination e-mail address.

uuencode

The first method uses `uuencode` to convert the binary file to flat text. This method works, but some e-mail clients, for instance my web-based SquirrelMail client, will not recognize the encoded file. If that is the case, you can simply save the text and decode it yourself with `uudecode` to obtain the original binary file.

First we define a variable specifying the temporary file that will contain the encoded message as well as a variable that holds the script-usage string.

```
#!/bin/sh
tmpfile=/tmp/uu_output.$$
usage="Usage: $0 {filename} {email_address}"
```

Now we validate the input by determining whether the positional parameter holding the binary file is defined.

```
if [ ! -z $1 ]
then
  file=$1
```


www.itbook.store/books/9781430218418

```
  if [ ! -f $file ]
  then
    echo $usage
    exit 1
```

If it is, we assign the `file` variable to its value. This is the file that will be encoded later in the script. If the parameter is not defined, we output the script usage and exit. Please note that the `exit` command is used with a value that will be the return code of the script. In this case it is nonzero (1), indicating that there was an issue encountered during execution.

Next you have to validate the positional parameter for the e-mail address that is passed to the script. This is done in the same way as for the file argument.

```
  else
    if [ ! -z $2 ]
    then
      address=$2
    else
      echo $usage
      exit 1
    fi
  fi
```

Finally, if no parameters were passed to the script, we display the script usage and exit with the nonzero return code as before.

```
else
  echo $usage
  exit 1
fi
```

Next is the heart of the script; it is the part that encodes the file and then sends the file to its destination.

```
basefile=`basename $file`
echo "A uuencoded file is attached called: $basefile" > $tmpfile
echo >> $tmpfile
uuencode $file $file >> $tmpfile
mail -s "$basefile attached from $from" $address < $tmpfile
rm $tmpfile
```

First a `tmpfile` is created with the `echo` command containing a summary of what is attached for the reader of the message. A blank line is added using the `echo` command between the text of the e-mail and the text of the encoded binary file. The file is encoded with the `uuencode` utility and appended to the `tmpfile` following the text of the message. The `uuencode` and `uudecode` commands were originally designed for this very purpose, to transmit a binary file via a transmission method that supports only text. After the file has

www.itbook.store/books/9781430218418

been encoded, we send the `tmpfile` to the destination e-mail address with the `mail` command; the `tmpfile` is then removed.

■ **Tip** If the `uuencode` and `uudecode` commands are not installed on your system, you can find the appropriate installation package of the UNIX `sharutils` utilities, where both commands are included. A version of `sharutils` should be available for most current platforms.

MIME Encoding

The next script performs roughly the same task as the first, but it uses MIME encoding. This type of encoding is more current than the `uuencode` method; it is also standard for e-mail clients that have to be able to process a message with an attachment.

The code here is very similar to the code used in the previous example except that it doesn't create an encoded file that is e-mailed; instead the script creates a specially formatted file constituting an e-mail message that contains the appropriate headers for e-mail clients to process and recognize as an attachment.

Once again we first define the temporary file that contains our message and the usage statement.

```
#!/bin/sh
tmpfile=/tmp/mime_output.$$
from="$USER@`cat /etc/dnsdomainname`"
usage="Usage: $0 {filename} {email address}"
```

The difference here is that we need to identify the message sender and add the sender's identity to the mail message manually.

Validation of the parameters passed to the script is the same as in the `uuencode` script.

```
if [ ! -z $1 ]
then
  file=$1
  if [ ! -f $file ]
  then
    echo $usage
    exit 1
  else
    if [ ! -z $2 ]
    then
      address=$2
    else
      echo $usage
      exit 1
```

www.itbook.store/books/9781430218418

```
    fi
  fi
else
  echo $usage
  exit 1
fi
```

Next the e-mail message is created, which includes a simple text message specifying which file is attached.

```
basefile=`basename $file`
cat > $tmpfile << EOT
From: $from
To: $address
Subject: $file attached from $from
MIME-Version: 1.0
Content-Type: multipart/mixed;boundary="mime-attachment-boundary"
--mime-attachment-boundary
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: 8bit
A MIME encoded file is attached called: $basefile
--mime-attachment-boundary
Content-Type: application/octet-stream; name="$basefile"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="$basefile"
EOT
```

In determining how to create the headers, I used my e-mail client to attach a binary file to a basic text message, and then sent it to myself. When I received it, I copied the headers into the test message. I have preserved the line spacing of the copied message.

Notice the line in the preceding code that defines a *boundary* string (`--mime-attachment-boundary`). This bounds the beginning and end of the text as well as the MIME-encoded portions of the message.

Once we've created the headers, we encode the binary file using the `mimencode` utility and then append the encoded file, which includes another boundary statement, to the temporary file to complete the message.

```
mimencode $file >> $tmpfile
echo --mime-attachment-boundary-- >> $tmpfile
```

Now that the temporary file containing the message is ready, we send it with the `sendmail` program using the `-t` option.

```
/usr/lib/sendmail -t < $tmpfile
rm $tmpfile
```

This option tells `sendmail` to look in the input for the `To:` header instead of specifying the destination address manually. Finally we remove the temporary file for cleanup.

www.itbook.store/books/9781430218418

■ **Tip** If the `mimencode` utility is not installed on your system, find it by installing the `metamail` package, which includes this utility.

One modification that could upgrade either of these two scripts would be to reverse the order of the input parameters so that the destination e-mail address comes first. That way you could accept multiple files and attach them all to a single message. With the `uuencode` version, it would simply be a matter of adding whitespace between the text segment for each encoded file. The `mimencode` version would be a bit more complex. You would need to separate the encoded sections with the appropriate boundary strings to signify the beginning and end of each attachment. To see exactly how this is done, send yourself a test message containing a couple of small attachments, and open the message with a text viewer instead of an e-mail client. This will allow you to see how the mail-message syntax is constructed for both the message headers as well as the embedded attachments.

www.itbook.store/books/9781430218418

www.itbook.store/books/9781430218418