10.1. Manipulating Strings

Bash supports a surprising number of string manipulation operations. Unfortunately, these tools lack a unified focus. Some are a subset of parameter substitution, and others fall under the functionality of the UNIX expr command. This results in inconsistent command syntax and overlap of functionality, not to mention confusion.

String Length

${#string}

expr length $string

These are the equivalent of *strlen()* in *C*.

expr "$string" : '.*'

```
stringZ=abcABC123ABCabc

echo ${#stringZ}                 # 15
echo `expr length $stringZ`      # 15
echo `expr "$stringZ" : '.*'`    # 15
```

Example 10-1. Inserting a blank line between paragraphs in a text file

```
#!/bin/bash
# paragraph-space.sh
# Ver. 2.1, Reldate 29Jul12 [fixup]

# Inserts a blank line between paragraphs of a single-spaced text file.
# Usage: $0 <FILENAME

MINLEN=60        # Change this value? It's a judgment call.
#  Assume lines shorter than $MINLEN characters ending in a period
#+ terminate a paragraph. See exercises below.

while read line  # For as many lines as the input file has ...
do
  echo "$line"   # Output the line itself.

  len=${#line}
  if [[ "$len" -lt "$MINLEN" && "$line" =~ [*{\.}]$ ]]
# if [[ "$len" -lt "$MINLEN" && "$line" =~ \[*\.\] ]]
# An update to Bash broke the previous version of this script. Ouch!
# Thank you, Halim Srama, for pointing this out and suggesting a fix.
    then echo    #  Add a blank line immediately
  fi             #+ after a short line terminated by a period.
done
```

```
exit

# Exercises:
# ---------
#  1) The script usually inserts a blank line at the end
#+    of the target file. Fix this.
#  2) Line 17 only considers periods as sentence terminators.
#     Modify this to include other common end-of-sentence characters,
#+    such as ?, !, and ".
```

Length of Matching Substring at Beginning of String

expr match "$string" '$substring'

> *$substring* is a regular expression.

expr "$string" : '$substring'

> *$substring* is a regular expression.

```
stringZ=abcABC123ABCabc
#       |------|
#       12345678

echo `expr match "$stringZ" 'abc[A-Z]*.2'`   # 8
echo `expr "$stringZ" : 'abc[A-Z]*.2'`       # 8
```

Index

expr index $string $substring

> Numerical position in $string of first character in $substring that matches.

```
stringZ=abcABC123ABCabc
#       123456 ...
echo `expr index "$stringZ" C12`             # 6
                                             # C position.

echo `expr index "$stringZ" 1c`              # 3
# 'c' (in #3 position) matches before '1'.
```

> This is the near equivalent of *strchr()* in *C*.

Substring Extraction

${string:position}

> Extracts substring from *$string* at *$position*.
>
> If the `$string` parameter is "*" or "@", then this extracts the positional parameters, [1] starting at `$position`.

${string:position:length}

Extracts `$length` characters of substring from `$string` at `$position`.

```
stringZ=abcABC123ABCabc
#       0123456789.....
#       0-based indexing.

echo ${stringZ:0}                            # abcABC123ABCabc
echo ${stringZ:1}                            # bcABC123ABCabc
echo ${stringZ:7}                            # 23ABCabc

echo ${stringZ:7:3}                          # 23A
                                             # Three characters of substring.



# Is it possible to index from the right end of the string?

echo ${stringZ:-4}                           # abcABC123ABCabc
# Defaults to full string, as in ${parameter:-default}.
# However . . .

echo ${stringZ:(-4)}                         # Cabc
echo ${stringZ: -4}                          # Cabc
# Now, it works.
# Parentheses or added space "escape" the position parameter.

# Thank you, Dan Jacobson, for pointing this out.
```

The *position* and *length* arguments can be "parameterized," that is, represented as a variable, rather than as a numerical constant.

Example 10-2. Generating an 8-character "random" string

```
#!/bin/bash
# rand-string.sh
# Generating an 8-character "random" string.

if [ -n "$1" ]  #  If command-line argument present,
then            #+ then set start-string to it.
  str0="$1"
else            #  Else use PID of script as start-string.
  str0="$$"
fi

POS=2  # Starting from position 2 in the string.
LEN=8  # Extract eight characters.

str1=$( echo "$str0" | md5sum | md5sum )
#  Doubly scramble     ^^^^^^   ^^^^^^
#+ by piping and repiping to md5sum.

randstring="${str1:$POS:$LEN}"
# Can parameterize ^^^^ ^^^^

echo "$randstring"
```

```
exit $?

# bozo$ ./rand-string.sh my-password
# 1bdd88c4

#  No, this is is not recommended
#+ as a method of generating hack-proof passwords.
```

If the `$string` parameter is "*" or "@", then this extracts a maximum of `$length` positional parameters, starting at `$position`.

```
echo ${*:2}          # Echoes second and following positional parameters.
echo ${@:2}          # Same as above.

echo ${*:2:3}        # Echoes three positional parameters, starting at second.
```

expr substr $string $position $length

Extracts *`$length`* characters from *`$string`* starting at *`$position`*.

```
stringZ=abcABC123ABCabc
#       123456789......
#       1-based indexing.

echo `expr substr $stringZ 1 2`              # ab
echo `expr substr $stringZ 4 3`              # ABC
```

expr match "$string" '\($substring\)'

Extracts *`$substring`* at beginning of *`$string`*, where *`$substring`* is a regular expression.

expr "$string" : '\($substring\)'

Extracts *`$substring`* at beginning of *`$string`*, where *`$substring`* is a regular expression.

```
stringZ=abcABC123ABCabc
#       =======

echo `expr match "$stringZ" '\(.[b-c]*[A-Z]..[0-9]\)'`   # abcABC1
echo `expr "$stringZ" : '\(.[b-c]*[A-Z]..[0-9]\)'`       # abcABC1
echo `expr "$stringZ" : '\(.......\)'`                   # abcABC1
# All of the above forms give an identical result.
```

expr match "$string" '.*\($substring\)'

Extracts *`$substring`* at *end* of *`$string`*, where *`$substring`* is a regular expression.

expr "$string" : '.*\($substring\)'

Extracts *`$substring`* at *end* of *`$string`*, where *`$substring`* is a regular expression.

```
stringZ=abcABC123ABCabc
```

```
#                ======

echo `expr match "$stringZ" '.*\([A-C][A-C][A-C][a-c]*\)'`    # ABCabc
echo `expr "$stringZ" : '.*\(......\)'`                       # ABCabc
```

Substring Removal

${string#substring}

Deletes shortest match of `$substring` from *front* of `$string`.

${string##substring}

Deletes longest match of `$substring` from *front* of `$string`.

```
stringZ=abcABC123ABCabc
#       |----|          shortest
#       |----------|    longest

echo ${stringZ#a*C}      # 123ABCabc
# Strip out shortest match between 'a' and 'C'.

echo ${stringZ##a*C}     # abc
# Strip out longest match between 'a' and 'C'.



# You can parameterize the substrings.

X='a*C'

echo ${stringZ#$X}      # 123ABCabc
echo ${stringZ##$X}     # abc
                        # As above.
```

${string%substring}

Deletes shortest match of `$substring` from *back* of `$string`.

For example:

```
# Rename all filenames in $PWD with "TXT" suffix to a "txt" suffix.
# For example, "file1.TXT" becomes "file1.txt" . . .

SUFF=TXT
suff=txt

for i in $(ls *.$SUFF)
do
  mv -f $i ${i%.$SUFF}.$suff
  #  Leave unchanged everything *except* the shortest pattern match
  #+ starting from the right-hand-side of the variable $i . . .
done ### This could be condensed into a "one-liner" if desired.

# Thank you, Rory Winston.
```

${string%%substring}

Deletes longest match of *$substring* from *back* of *$string*.

```
stringZ=abcABC123ABCabc
#                    ||     shortest
#        |------------|     longest

echo ${stringZ%b*c}      # abcABC123ABCa
# Strip out shortest match between 'b' and 'c', from back of $stringZ.

echo ${stringZ%%b*c}     # a
# Strip out longest match between 'b' and 'c', from back of $stringZ.
```

This operator is useful for generating filenames.

Example 10-3. Converting graphic file formats, with filename change

```
#!/bin/bash
#  cvt.sh:
#  Converts all the MacPaint image files in a directory to "pbm" format.

#  Uses the "macptopbm" binary from the "netpbm" package,
#+ which is maintained by Brian Henderson (bryanh@giraffe-data.com).
#  Netpbm is a standard part of most Linux distros.

OPERATION=macptopbm
SUFFIX=pbm          # New filename suffix.

if [ -n "$1" ]
then
  directory=$1      # If directory name given as a script argument...
else
  directory=$PWD    # Otherwise use current working directory.
fi

#  Assumes all files in the target directory are MacPaint image files,
#+ with a ".mac" filename suffix.

for file in $directory/*    # Filename globbing.
do
  filename=${file%.*c}      #  Strip ".mac" suffix off filename
                            #+ ('.*c' matches everything
                            #+ between '.' and 'c', inclusive).
  $OPERATION $file > "$filename.$SUFFIX"
                            # Redirect conversion to new filename.
  rm -f $file               # Delete original files after converting.
  echo "$filename.$SUFFIX"  # Log what is happening to stdout.
done

exit 0

# Exercise:
# --------
#  As it stands, this script converts *all* the files in the current
#+ working directory.
#  Modify it to work *only* on files with a ".mac" suffix.
```

Example 10-4. Converting streaming audio files to *ogg*

```
#!/bin/bash
# ra2ogg.sh: Convert streaming audio files (*.ra) to ogg.

# Uses the "mplayer" media player program:
#      http://www.mplayerhq.hu/homepage
# Uses the "ogg" library and "oggenc":
#      http://www.xiph.org/
#
# This script may need appropriate codecs installed, such as sipr.so ...
# Possibly also the compat-libstdc++ package.


OFILEPREF=${1%%ra}      # Strip off the "ra" suffix.
OFILESUFF=wav           # Suffix for wav file.
OUTFILE="$OFILEPREF""$OFILESUFF"
E_NOARGS=85

if [ -z "$1" ]          # Must specify a filename to convert.
then
  echo "Usage: `basename $0` [filename]"
  exit $E_NOARGS
fi


##########################################################################
mplayer "$1" -ao pcm:file=$OUTFILE
oggenc "$OUTFILE"  # Correct file extension automatically added by oggenc.
##########################################################################

rm "$OUTFILE"      # Delete intermediate *.wav file.
                   # If you want to keep it, comment out above line.

exit $?

#  Note:
#  ----
#  On a Website, simply clicking on a *.ram streaming audio file
#+ usually only downloads the URL of the actual *.ra audio file.
#  You can then use "wget" or something similar
#+ to download the *.ra file itself.


#  Exercises:
#  ---------
#  As is, this script converts only *.ra filenames.
#  Add flexibility by permitting use of *.ram and other filenames.
#
#  If you're really ambitious, expand the script
#+ to do automatic downloads and conversions of streaming audio files.
#  Given a URL, batch download streaming audio files (using "wget")
#+ and convert them on the fly.
```

A simple emulation of getopt using substring-extraction constructs.

Example 10-5. Emulating *getopt*

```
#!/bin/bash
# getopt-simple.sh
# Author: Chris Morgan
# Used in the ABS Guide with permission.


getopt_simple()
{
    echo "getopt_simple()"
    echo "Parameters are '$*'"
    until [ -z "$1" ]
    do
      echo "Processing parameter of: '$1'"
      if [ ${1:0:1} = '/' ]
      then
          tmp=${1:1}               # Strip off leading '/' . . .
          parameter=${tmp%%=*}     # Extract name.
          value=${tmp##*=}         # Extract value.
          echo "Parameter: '$parameter', value: '$value'"
          eval $parameter=$value
      fi
      shift
    done
}

# Pass all options to getopt_simple().
getopt_simple $*

echo "test is '$test'"
echo "test2 is '$test2'"

exit 0  # See also, UseGetOpt.sh, a modified version of this script.

---

sh getopt_example.sh /test=value1 /test2=value2

Parameters are '/test=value1 /test2=value2'
Processing parameter of: '/test=value1'
Parameter: 'test', value: 'value1'
Processing parameter of: '/test2=value2'
Parameter: 'test2', value: 'value2'
test is 'value1'
test2 is 'value2'
```

Substring Replacement

${string/substring/replacement}

Replace first *match* of `$substring` with `$replacement`. [2]

${string//substring/replacement}

Replace all matches of `$substring` with `$replacement`.

```
stringZ=abcABC123ABCabc

echo ${stringZ/abc/xyz}       # xyzABC123ABCabc
                              # Replaces first match of 'abc' with 'xyz'.

echo ${stringZ//abc/xyz}      # xyzABC123ABCxyz
                              # Replaces all matches of 'abc' with # 'xyz'.

echo  ---------------
echo "$stringZ"               # abcABC123ABCabc
echo  ---------------
                              # The string itself is not altered!

# Can the match and replacement strings be parameterized?
match=abc
repl=000
echo ${stringZ/$match/$repl}  # 000ABC123ABCabc
#              ^      ^         ^^^
echo ${stringZ//$match/$repl} # 000ABC123ABC000
# Yes!          ^      ^        ^^^         ^^^

echo

# What happens if no $replacement string is supplied?
echo ${stringZ/abc}           # ABC123ABCabc
echo ${stringZ//abc}          # ABC123ABC
# A simple deletion takes place.
```

${string/#substring/replacement}

If `$substring` matches *front* end of `$string`, substitute `$replacement` for `$substring`.

${string/%substring/replacement}

If `$substring` matches *back* end of `$string`, substitute `$replacement` for `$substring`.

```
stringZ=abcABC123ABCabc

echo ${stringZ/#abc/XYZ}          # XYZABC123ABCabc
                                  # Replaces front-end match of 'abc' with 'XYZ'.

echo ${stringZ/%abc/XYZ}          # abcABC123ABCXYZ
                                  # Replaces back-end match of 'abc' with 'XYZ'.
```

10.1.1. Manipulating strings using awk

A Bash script may invoke the string manipulation facilities of awk as an alternative to using its built-in operations.

Example 10-6. Alternate ways of extracting and locating substrings

```
#!/bin/bash
# substring-extraction.sh
```

```
String=23skidoo1
#      012345678    Bash
#      123456789    awk
# Note different string indexing system:
# Bash numbers first character of string as 0.
# Awk  numbers first character of string as 1.

echo ${String:2:4} # position 3 (0-1-2), 4 characters long
                                         # skid

# The awk equivalent of ${string:pos:length} is substr(string,pos,length).
echo | awk '
{ print substr("'"${String}"'",3,4)      # skid
}
'
#  Piping an empty "echo" to awk gives it dummy input,
#+ and thus makes it unnecessary to supply a filename.

echo "----"

# And likewise:

echo | awk '
{ print index("'"${String}"'", "skid")      # 3
}                                           # (skid starts at position 3)
'   # The awk equivalent of "expr index" ...

exit 0
```

10.1.2. Further Reference

For more on string manipulation in scripts, refer to Section 10.2 and the relevant section of the expr command listing.

Script examples:

1. Example 16-9
2. Example 10-9
3. Example 10-10
4. Example 10-11
5. Example 10-13
6. Example A-36
7. Example A-41

Notes

[1] This applies to either command-line arguments or parameters passed to a function.

[2] Note that `$substring` and `$replacement` may refer to either *literal strings* or *variables*, depending on context. See the first usage example.

Prev Home Next
Manipulating Variables Up Parameter Substitution