Pattern Matching and Substitution in `bash`

R (Chandra) Chandrasekhar

2023-02-28 | 2023-03-01

Parsing filenames

A fully qualified filename consists of a `path`, a `basename`, and a file `extension`. While not all filenames are encountered in their full glory, it helps to decompose any given filename into its constituent parts to help with housekeeping functions on a machine running `bash`—for example, to facilitate searching, sorting and other file-related functions.

Extended globbing

Globbing is the unflattering term—abbreviation for *global*—used to denote an operation to extract files satisfying certain conditions [1]. It is applicable also to the `bash` command line. For our purposes, it is useful and sometimes mandatory to set `shopt -s extglob` after the shebang line.

A canonical filename

A canonical filename will comprise these components:

a. a *path* with the forward slash `/` as the separator between elements denoting the path;
b. a filename comprising a *basename* which appears immediately after the *last* `/` character;
c. a *file extension* that occurs after the basename immediately after a `.` or period character.

`/my_path/is/quite/long/basename.ext` is a canonical filename where the abovenamed elements are as follows:

1. path: `/my_path/is/quite/long/`
2. basename: `basename`
3. extension: `ext`
4. filename: `basename.ext`

Parsing the filename

Our next task is to dissect the canonical filename into its above components using `bashisms`:

```
#!/bin/bash
shopt -s extglob

fullname="/my_path/is/quite/long/basename.ext"
echo "fullname is ${fullname}"

#
# Extract path
# Approach from the right until the _first_ `/` is encountered
# and throw away everything from the _right_ end up to and including that `/`.
#
path="${fullname%/*}"
echo "path is ${fullname%/*}"

#
# Extract filename
# Approach from the left until the _last_+ `/` character is encountered
# and throw away everything from the _left_ end up to and including that `/`.
#
filename="${fullname##*/}"
echo "filename is ${fullname##*/}"

#
# Extract file extension
# Approach from the _left_ until the _last_ `.` character is encountered
# and throw away everything from the _left_ up to and including that last `.`.
#
ext="${fullname##*.}"
echo "extension is ${fullname##*.}"

#
# Extract basename
# This requires trimming strings from both the left and the right of `fullname`
# and requires _two_ steps.
#
# Instead, we use `filename` which is already available, and excise the extension.
# For this, we approach from the _right_ until we encounter the _first_ `.` character
# and throw away everything from the _right_ up to and including that last `.`.
#
basename="${filename%.*}"
echo "basename is ${filename%.*}"
```

Mnemonics behind the # and % symbols

The use of the symbols # and % in the pattern matching expressions might seem arbitrary or whimsical. For a start, they do not conform to the usual delimiters ^ and $ for the beginning and end of a line. Because we are not line-oriented here, those symbols are not used.

One other point to keep in view constantly is t avoid looking at bash pattern matching through the lens of regular expressions. There are some similarities but the two are not identical.

So, what's the dope on # and %? These two symbols have been chosen for their near universal usage a prefix and suffix respectively. It is customary to write #1 for "number one" and 20% for "twenty percent" where you will notice that the # is written as a *prefix* and the % is written as a *suffix* to the number.

In the bash pattern-matching we have encountered so far, we are matching elements in a string and throwing away the matching portion, using some known delimiter. When we match from the left, we use # because it is a prefix. Likewise, when we match from the right, we use %, which is a suffix. In both cases, we stop at the first match from whichever direction we are starting the match. The single # and % therefore denote lazy matching.

The symbol ## means we deal with the *longest* substring from the left that matches: a case of greedy matching.The same applies to %%.

If you look carefully, you will see that what we do not care about what we are throwing away. We can therefore refer to it with the * character, which denotes one or more characters, whether as a wildcard or a glob. What *is* important to us, though, is the *delimiter* that anchors the string that we are trimming off.

This delimiting character will be placed to the right of the * when used with # or ##, and it will be placed to the left of * when used with % or %%. You will notice that the / and . characters obey this simple, logical placement rule in the code above. In both cases, the anchoring delimiter is also trimmed off.

Acknowledgements

Feedback

Please email me your comments and corrections.

A PDF version of this article is available for download here:

https://swanlotus.netlify.app/blogs/pattern-matching-in-bash.pdf

References

[1] Wikipedia, 'Glob (programming)', 15-Jan-2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Glob_(programming)&oldid=1133836865. [Accessed: 28-Feb-2023]

