Extending shell conditionals (DRAFT)
2011-11-15

This white paper proposes various extensions to the existing POSIX shell conditional statements. It supports Austin group defect report 375 (http://austingroupbugs.net/view.php?id=375), in particular reply 967.

This is a ***draft***. As it is spread more widely, it is expected to change. The intent is to provide a single location where the issues can be discussed in an organized fashion.

Table of contents:

Overview

Issue

Many implementations of "test" (aka "["), including shell built-ins, implement conditionals beyond those specified in the current version of POSIX. What's more, many extant programs rely on these extensions. This proposal recommends formally adding these widely-implemented extensions to the POSIX specification itself, as these extensions have become widespread and are ready to be standardized. Each of these additions is described separately, since they can be treated separately.

Background

Austin group defect report 375 (http://austingroupbugs.net/view.php?id=375) ("Extend test/ [...] conditionals: ==, &lt;, &gt;, -nt, -ot, -ef") proposed extensions to POSIX "test". It proposed adding certain widely-implemented and widely-used extensions of "test" to the POSIX standard.

This defect report was discussed at the September 8, 2011 teleconference meeting and it was agreed that the submitter should "produce a whitepaper expanding the proposal (similar to proposals made in the past, for example the LFS proposal, http://www.unix.org/version2/whatsnew/lfs20mar.html (Adding Support for Arbitrary File Sizes to the Single UNIX Specification). This could then be widely circulated amongst all interested parties to look for consensus. The standard developers recommend that the white paper should pay particular attention to note 670."

This document is the whitepaper requested by the Austin group. This whitepaper attempts to expand the proposal so that it can be "widely circulated amongst all interested parties to look for consensus". It attempts to respond to standards developers recommendations, in particular, to pay "particular attention to note 670."

Requirements

These proposals are only proposed because they meet the following requirements:

1. Are already implemented in at least one implementation.
2. Are used in existing programs/scripts.
3. Are easily implemented.

Importance

All of these proposals can be implemented in other ways, but their omission in POSIX can render otherwise-compatible scripts non-conforming. Some of these extensions are identified as "bashisms" in pages such as http://mywiki.wooledge.org/Bashism, but in fact these are widely implemented and/or depended upon, whether or not bash is used.

Their widespread use and implementation suggests that they are ready to be added to POSIX itself.

Proposed changes to the POSIX specification

Add “==” to test

In the text of test, circa page 3224, add the following primary definition:

- s1 == s2 True if the strings s1 and s2 are identical; otherwise, false. This primary is equivalent to s1 = s2.

Add “-nt” (newer-than) and “-ot” (older-than) to test

In the text of test, circa page 3224, add the following primary definition:

- pathname1 -nt pathname2 True if pathname1 exists and pathname2 does not, or if both exist and pathname1 is newer than pathname2 according to their modification times; otherwise, false.
- pathname1 -ot pathname2 True if pathname2 exists and pathname1 does not, or if both exist and pathname1 is older than pathname2 according to their modification times; otherwise, false.

Add “-ef” to test

In the text of test, circa page 3224, add the following primary definition:

- "pathname1 -ef pathname2". True if pathname1 and pathname2 name the same file, otherwise, the result is false.

Add “[[”

In XCU section 2.4, line 72478, add “[[“ and “]]” to the list of reserved words. On line 72491, remove “[[“ and “]]” from the reserved word list.

In section 2.6, after line 72671, add: “Expressions directly enclosed by [[and]] do not perform field splitting or pathname expansion; see the section on double bracket expressions for more information.”

In section 2.9.4.1 (“Grouping Commands” inside “Compound Commands”), page 2321, add the following grouping command:
[[expression]]   Execute the expression in the current process environment, and return a status of 0 or 1. The expression is evaluated using the rules of DOUBLE BRACKET EXPRESSIONS.

Later in the text, add the following:

DOUBLE BRACKET EXPRESSIONS

Inside [[…]], expressions are evaluated to return a status of 0 or 1. Field splitting and pathname expansion are *not* performed on the words directly enclosed between the [[and]], but other expansions are performed: tilde expansion, quote removal, parameter expansion, command substitution, and arithmetic expansion, as described in section 2.6 (word expansions). Conditional operators *must* be unquoted to be recognized as conditional operators; if they are quoted, they are not considered conditional operators inside double bracket expressions.

All of the conditional operators of test are available, with the following exceptions and additions:

- There must be at least two parameters. The single-parameter form (which was a shorthand for determining if a string is non-empty) is not accepted (use "-n" instead).
- Support for the **-a** (and) and **-o** (or) operators are not required. They are replaced with **&&** and **||**, as described below.
- "string == pattern". Returns true if string matches the pattern, as described in "Pattern Matching Notation" (section 2.13). Otherwise, return false.
- "string != pattern". Return the logical negation of "string == pattern".
- "string = pattern". Not defined in this specification; implementations are free to implement it the same as "==".
- "string =~ regex". Return true (0) if string matches the extended regular expression regex, otherwise return false (1).
- "string1 < string2". Compare two strings lexicographically, using the current locale settings, and return true if string1 is less than string2.
- "string1 > string2". Compare two strings lexicographically, using the current locale settings, and return true if string1 is greater than string2.

Expressions may be combined as follows (in decreasing order of precedence):

- "(e)". Returns the value of expression e.
- "! e". Returns true (0) if expression e is false, else returns false (1).
- "e1 && e2". Returns true if both e1 and e2 are true, else returns false. This is a short-circuit evaluation; if e1 is false, e2 is not evaluated.
- "e1 || e2". Returns true if either e1 or e2 are true, else returns false. This is a short-circuit evaluation; if e1 is true, e2 is not evaluated.

In the grammar of section 2.10.2, after line 73492, add the following:

After 73505 (CLOBBER) and its comment, add:

```
%token DLBRACKET  DRBRACKET
/* '[[' ']]' */
```

After line 73545 ("| subshell"), add:

```
  | double_bracket_expression
```

After 73553 (definition of subshell), add:

```
 double_bracket_expression : DLBRACKET inner_db_expression DRBRACKET ;
```

Rationale

Add “==” to test

This proposed change adds primary "s1 == s2" as a synonym for "s1 = s2".

There are three major reasons for adding “==”:

1. *Primary “==” is more visually distinct from assignment (“=”).* Since “=” is also used for assignment in shell scripts, using “==” for “is equals” makes the comparison visually distinctive, making it clearer to readers that “is equals” is intended.
2. *Allowing “==” for “is-equal-to” adds consistency with other programming languages that use “=” for assignment.* Most languages that use “=” for assignment also use “==” for “is equals” so that these operations are more visually distinct. These include C, C++, Java, C#, Python, and Perl. It is oddly inconsistent that test/[do not support “==” as well. Many languages (like Pascal) that use "=" for comparison use another spelling (like ":=") for assignment, again, to keep their spellings separate. In some cases these languages can always disambiguate from context, and even then, they intentionally do not use the same spelling. It's too late to get rid of "=" for comparison, but it's easy to add "==" as a synonym, which is what is proposed.
3. *Primary “==” is already widely implemented in many implementations and is used in many shell scripts.* This suggests that there is value in standardizing it.

A counter-argument to adding “==” is that it is redundant with “=”. This is true, but there are many other redundancies in POSIX. For example, “[” is redundant with “test” but this is not considered a problem. In any case, it is a redundancy that is considered valuable by many; “=” came first, and implementers have added “==” since.

It could be argued that perhaps “==” should mean “numeric equality” instead of “string equality”. However, “==” is already widely implemented and used for string equality, and there are no instances where it is implemented or used to mean numeric equality in a test/[implementation. This universal agreement suggests that string equality is the right semantic to standardize.

Obviously, there is no *requirement* that assignment and is-equal-to, be visually distinctive, since they are disambiguated by being part of test/[or not. There is no technical ambiguity, since “=” only means “is-equal” inside test/[. But the many shell and test implementations which do support “==”; their many users, and the many other languages which do this, suggest that this is widely considered to be useful.

This proposal is *not* a proposal that “everyone just switch to bash”. This is widely-implemented extension, not one implemented solely by bash.

The primary “==” is a very widely-implemented synonym for “=” and in all cases it is implemented as a synonym for “=”. The “==” in test is already implemented in the following implementations:

1. GNU bash: Supports ==.
2. GNU coreutils “test”: Added "==" support on 2011-03-22.
3. ash: Supports ==.
4. pdksh (public domain korn shell): Supports ==, see http://web.cs.mun.ca/~michael/pdksh/pdksh-man.html (Note that some system’s “ksh” is actually pdksh).
5. mksh (MirBSD(TM) Korn Shell): Supports ==. See http://www.mirbsd.org/mksh.htm
6. OpenBSD's /bin/sh: Supports "==" (it's not documented, but it DOES work).
7. FreeBSD-current's /bin/sh and /bin/test have recently added "==". See http://svn.freebsd.org/base/head/bin/test/test.c.
8. busybox ash: Supports ==. This is particularly remarkable, since busybox is designed for relatively small systems and emphasizes small code size. Yet even busybox implements “==”.

A few implementations do not support “==”, but even in those cases it tends to be trivial to add:

1. NetBSD’s sh doesn't support "==", but a patch has been submitted to add it. The last comment (2011-03-18) on it was positive, but is is not clear what they will do with it: http://gnats.netbsd.org/cgi-bin/query-pr-single.pl?number=44733. However, if this is added to POSIX, it is likely to be added to NetBSD sh.
2. The dash shell does not support "==", but doing so is a one-line patch. This patch is at http://permalink.gmane.org/gmane.comp.shells.dash/498 and was submitted on 2011-03-06. The developers seemed to agree that if POSIX added "==" as a requirement, dash would implement it.

This proposal has no effect on the official ksh from AT&T; ksh doesn't have test/[built-in, so it simply uses the underlying implementation of test/[. . Note that some systems have a “ksh” that is actually a pdksh. AT&T ksh does have “[[“ and inside this it does support “==” as a synonym for “=”; in fact, it considers “=” obsolete inside “[[“.

Add “-nt” (newer-than) and “-ot” (older-than) to test

This proposal adds primaries -nt (newer-than) and -ot (older-than) for comparing modification times. Determining if something should be done, based on whether or not one file is newer than another, is a common operation. Thus, it makes sense to include the ability to easily compare modification times of filesystem objects.

It is possible to get the same effect using the standard mechanisms using awkward expressions such as ["$(find 'pathname1' -prune -newer 'pathname2')"]. However, this is not at all clear, and is much more complicated. This extension is widely implemented, and this proposal adds it the standard.

One challenge is that there is some disagreement on what the semantics should be if files do not exist. Possibilities for the standard are:

1. *Both files must exist for a "true" result.* This is the semantic of "dash" and some other implementations. This can be expressed as:
    a. pathname1 −nt pathname2:True if pathname1 and pathname2 exist and pathname1 is newer than pathname2 according to their modification times; otherwise false.
    b. pathname1 −ot pathname2: True if pathname1 and pathname2 exist and pathname1 is older than pathname2 according to their modification times; otherwise false.
2. *A nonexistent file is considered older than a file that does exist.* This is the semantic of bash and current pdksh. Note that pdksh version 5.2.14 *switched* to this semantic in 1999, suggesting that there was value to this particular semantic. This semantic can be expressed as:
    a. pathname1 -nt pathname2: True if pathname1 exists and pathname2 does not, or if both exist and pathname1 is newer than pathname2 according to their modification times; otherwise, false. (Note that if pathname1 does not exist, the result is false.)
    b. pathname1 -ot pathname2: True if pathname2 exists and pathname1 does not, or if both exist and pathname1 is older than pathname2 according to their modification times; otherwise, false. (Note that if pathname2 does not exist, the result is false.)
3. Allow either semantic. An example would be:
    a. pathname1 −nt pathname2:True if both pathname1 and pathname2 exist and pathname1 is newer than pathname2 according to their modification times. False if pathname1 does not exist. Otherwise, it is unspecified if it returns true or false.
    b. pathname1 −ot pathname2: True if both pathname1 and pathname2 exist and pathname1 is older than pathname2 according to their modification times. False if pathname2 does not exist. Otherwise, it is unspecified if it returns true or false.

An argument for option 1 is that its description is slightly simpler. But it is not much simpler.

The proposal here recommends option 2, namely, that nonexistent files be considered older, for the following reasons:

1. This makes it simple to express the case where a file "overrides" an older file, as a file that exists is considered newer than a file that does not exist.
2. Tighter semantics are in general desirable, where practical.
3. Since pdksh intentionally switched to this semantic, this suggests that this is a more useful semantic.

An argument for option 3 is that no one has to change their implementation to match. If option 2 is not accepted, option 3 would be a reasonable alternative, especially since there would always be the option to tighten up the semantics in some future version of POSIX if necessary.

Add “-ef” to test

In many cases it is useful to know if two different filenames refer to the same file. For example, http://gcc.gnu.org/bugzilla/show_bug.cgi?id=30838 reports on a shell script “gen-classlist.sh” with the following line, so that certain actions will only occur if two different directory names refer to different directories:

```
if test ! "${top_builddir}" -ef "@top_srcdir@"; then
```

This text is worded as “refer to the same file” instead of simply “are hard linked,” as this is what extant implementations actually do. In particular, if files symbolically link to the same eventual file, comparing them with “-ef” should produce “true”. Austin group defect report 375, reply 670, reports that:

```
touch a; ln -s a b; test a -ef b
```

sets $? to 0 on at least bash and GNU coreutils test (at least).

This primary is currently implemented in at least bash, busybox sh, and GNU coreutils test.

Add “[[”

The test/[operator can sometimes be difficult to use correctly. Word splitting and pathname expansion can require many quote characters to do simple comparisons. Longer expressions (involving “-a” or “-o”) can be misinterpreted. The “<” and “>” comparisons do not need to be quoted (and unless various test extensions are done according to the locale).

Perhaps most concerningly, it is not particularly simple to compare strings with various text patterns using test/[. Developers sometimes use “case” to compare variables with a pattern, simply because test does not include a mechanism for doing so. And “case” only supports the simple globbing scheme, not the far more capable regular expression pattern-matching mechanism. The “[[“ adds these pattern-matching capabilities for when it is needed.

The grammar given above stops at the point where “test” is no longer defined in a grammar.

The proposed semantics are based on the “[[“ implementations of bash (see http://www.gnu.org/s/bash/manual/html_node/Conditional-Constructs.html#Conditional-Constructs), pdksh (see http://web.cs.mun.ca/~michael/pdksh/pdksh-man.html) , and AT&T ksh93 (http://www2.research.att.com/sw/download/man/man1/ksh.html).

Alternative proposals

Adding “<” and “>” to test

Early versions of this proposal also proposed this:.

In the text of test, circa page 3224, add the following primary definitions:

- s1 < s2 True if the string s1 is lexicographically less than s2; otherwise, false.
- s1 > s2 True if the string s1 is lexicographically greater than s2; otherwise, false.

However, comment #670 by eblake on 2011-02-07 (see http://austingroupbugs.net/view.php?id=375 comment #670) made some good points about the problems with these primaries. He noted that < and > inside test/[must be quoted. Also, existing implementations often fail to implement locale-specific collation. Thus, as recommended by eblake, an effort has been made to standardize [[, where < and > do not have to be quoted, and where collation is done according to locale.

Appendix A: Interpretation of -nt and -ot

Unfortunately, there are differences in how -nt and -ot are implemented in different shells. This appendix shows the differences in detail, to help justify the options and the one selected above.

http://austingroupbugs.net/view.php?id=375 bugnote 975 includes a report from gber on 2011-09-25 stating: "It should be noted that there are widespread implementations of test -nt/-ot with different and incompatible semantics in FreeBSD/NetBSD/OpenBSD and dash. These test implementations all trace their roots to the test builtin of pdksh before version 5.2.14, the difference to the behavior described above is that test will return failure in case the second file does not exist. The test implementations with this behavior have been used by NetBSD since 1994 and by FreeBSD since 1999 and it seems to have been used by dash since the first Linux port of ash in 1993."

In particular, pdksh trunk changed its semantics in 1999 with this changelog entry from http://web.cs.mun.ca/~michael/pdksh/ChangeLog :

```
Wed Jun 30 17:42:54 NDT 1999 Michael Rendell (michael@lyman.cs.mun.ca)
        * c_test.c(test_eval): changed -nt/-ot tests so they succeed
          if pathname2 (pathname2) `does not exist' (ie, the stat fails).
          (based on fix from Dave Hillman).
```

To determine various systems' behavior, the following script was run in a directory with files "n" (newer) and "o" (older), and no such files named 1 or 2:

```
result() {
  if [ "$?" = 0 ] ; then
      echo "t"
  else
      echo "f"
  fi
}
```

```
# files 1 and 2 don't exit.  File "o" is older than file "n" (newer):
ITEMS="1 o n"

echo "Smoke test: Produce false and true:"
false ; result
true ; result

echo "test -nt, for files $ITEMS:"
for left in $ITEMS ; do
  for right in $ITEMS 2 ; do
      if ! [ "$right" = "2" ] || [ "$left" = "1" ] ; then
      printf "%s -nt %s: " "$left" "$right"
      test $left -nt $right ; result
      fi
  done
done

echo "test -ot, for files $ITEMS:"
for left in $ITEMS ; do
  for right in $ITEMS 2 ; do
      if ! [ "$right" = "2" ] || [ "$left" = "1" ] ; then
      printf "%s -ot %s: " "$left" "$right"
      test $left -ot $right ; result
      fi
  done
done
```

The following are produced by GNU bash 4.1.10(4), GNU coreutils test, and pdksh version 5.2.14:

```
Smoke test: Produce false and true:
f
t
test -nt, for files 1 o n:
1 -nt 1: f
1 -nt o: f
1 -nt n: f
1 -nt 2: f
o -nt 1: t
o -nt o: f
o -nt n: f
n -nt 1: t
n -nt o: t
```

```
n -nt n: f
test -ot, for files 1 o n:
1 -ot 1: f
1 -ot o: t
1 -ot n: t
1 -ot 2: f
o -ot 1: f
o -ot o: f
o -ot n: t
n -ot 1: f
n -ot o: f
n -ot n: f
```

The following is produced by dash version 0.5.6.1:

```
Smoke test: Produce false and true:
f
t
test -nt, for files 1 o n:
1 -nt 1: f
1 -nt o: f
1 -nt n: f
1 -nt 2: f
o -nt 1: f
o -nt o: f
o -nt n: f
n -nt 1: f
n -nt o: t
n -nt n: f
test -ot, for files 1 o n:
1 -ot 1: f
1 -ot o: f
1 -ot n: f
1 -ot 2: f
o -ot 1: f
o -ot o: f
o -ot n: t
n -ot 1: f
n -ot o: f
n -ot n: f
```

The output of “diff -u ,bash ,dash” is, briefly (the results of “bash” are shown with “-” while the results of dash are shown with “+”):

```
 1 -nt o: f
 1 -nt n: f
 1 -nt 2: f
-o -nt 1: t
+o -nt 1: f
 o -nt o: f
 o -nt n: f
-n -nt 1: t
+n -nt 1: f
 n -nt o: t
 n -nt n: f
 test -ot, for files 1 o n:
 1 -ot 1: f
-1 -ot o: t
-1 -ot n: t
+1 -ot o: f
+1 -ot n: f
 1 -ot 2: f
 o -ot 1: f
 o -ot o: f
```