Four UNIX Programs in Four UNIX Collections: Seeking Consistency in an Open Source Icon

Doug Thayer and Keith Miller
Dept. of Computer Science
University of Illinois at Springfield
miller.keith@uis.edu

Abstract

An important claim for Open Source Software (OSS) is that it will, over time, improve (Martin, 2003 and Cingely, 2003). Critics of OSS make an opposite claim: that OSS will tend to degrade because its development is unplanned and chaotic (McKendrick, 2003). Although there is much passion in these arguments, there is often little hard data presented to back up either of these claims.

In this paper, we present a modest amount of data on a leading Open Source initiative. Using N-version testing techniques (Grissom and Miller, 1999), we explore four UNIX utility programs in four different UNIX implementations in two separate releases. We then compare and contrast this software, examining their agreements and disagreements, the progress between the two versions, the lines of code, and their execution speed.

OSS proponents would like for UNIX utilities to, over time, converge to the same ("correct") behavior. Some of our results indicate a high degree of agreement among the four implementations for many UNIX calls, especially those that don't use optional parameters. However, there is more variation in the number of options supported by the four implementations. These variations were also visible in the amount of code required in the four different implementations. There was a wide range in the number of lines of code used to implement the utilities. For example, the lines of code to implement *grep* varied from a low of 167 to a high of 23141 lines.

The range in performance statistics was also striking. For example, for one popular utility program, *cat,* the performance measures ranged from a best of 5.4 milliseconds to a worst of 109.1 milliseconds. On an optimistic note, the worst performer in its first version improved its performance to 6.2 milliseconds in the second version tested.

Like so much in the messy world of empirical software engineering research, our results were mixed. However, our overall impression is that these UNIX utilities largely agreed on the correct behavior for those options that all implementations included. Their behavior (both the similarities and differences between the different implementations) appeared to be stable between the two versions.

Introduction

An important claim for Open Source Software (OSS) is that it will, over time, improve (Martin, 2003 and Cingely, 2003). Critics of OSS make an opposite claim, that OSS will tend to degrade because its development is unplanned and chaotic (McKendrick, 2003). Although there is much passion in these arguments, there is often little hard data presented to back up either of these claims.

In this paper, we present a modest amount of data on a leading Open Source initiative. Using N-version testing techniques (Grissom and Miller, 1999), we explore four UNIX utility programs in four different UNIX implementations in two separate releases. We then compare and contrast this software, examining their agreements and disagreements, the progress between the two versions, the lines of code, and their execution speed.

UNIX Utilities

The UNIX Utilities are simple programs used with the UNIX operating system to provide a basic level of functionality to users and administrators. There exists a broad diversity of many similar implementations, collections of programs that go by the same name and perform similar tasks, but were developed by different developer groups, sometimes for very different purposes using different implementation strategies and different programming languages.

The UNIX operating system has been evolving since 1975, and may persist for a long time hence. The time-honored UNIX Utilities will be part of this computing legacy; this justifies investment in improving reliability and performance of the UNIX Utilities.

Because of the modularity of the individual UNIX Utilities, it is within the reach of a single programmer working informally to replicate the programs one by one. Thus it is not surprising that there are currently multiple open-source implementations available for use with open-source UNIX variants. Whereas we cannot compare the UNIX Utilities implementations with a standardized requirements document, we can inter-compare the various implementations to discover where they differ in their behavior.

Previous Related Research

Previous experiments have discovered faults in several commercial implementations of the UNIX Utilities. (Miller et al., 1990) found program-crash errors when the utilities were subjected to random input streams. Miller et al. (1995) revisited the experiments five years later and found little to no improvement in the studied UNIX Utilities implementations.

In general, there is an issue as to whether reliability problems reflect genuine user concerns (likely inputs). The research in both studies cited above used random strings with an equally-weighted probability of occurrence of each character in the ASCII character set (0-255). Also there is the general issue of program control flow changes controlled by command line options. These studies tested only the default control flow of the tested utilities. The research reported in this paper extends this previous research by using a different set of tests, different implementations, and more recent versions than the 1990 and 1995 work.

N-Version Testing

Because the UNIX Utilities lack a standard specification document, the programs cannot be tested against their specifications. However, they can be tested against each other using "N-version testing," (Grissom and Miller, 1999) a technique based on N-version programming (Knight and Leveson, 1986). In N-version testing, all N versions (in this paper, four versions) are executed with the same input and the outputs are compared. Where the outputs match, the programs agree. Interesting cases arise where the outputs differ. In some cases it is a spurious element of program output (a usage message or program identification) but in some cases the difference reflects a genuine disagreement between at least two of the programs. A difference might reveal a fault (an unintended difference) in one of the implementations, or it might reveal a different interpretation of the desired functionality (an intentional difference). It is important to note that our testing procedure tests the sameness, not the correctness, of the programs being tested.

The Experiment

We chose four collections of OSS UNIX Utilities to test: the GNU utilities (http://www.gnu.org), Busybox (http://busybox.net/), asmutils (http://linuxassembly.org/asmutils.html), and Perl Power Tools (PPT) (http://www.perl.com/language/ppt/). All of the collections were available for download as of this writing. Table 1 lists the four collections, their intended use, and the programming language used in the implementation.

Table 1. The four collections of UNIX Utilities used in our experiment.

Collection	Intended Use	Language
asmutils	embedded assembler code	i386 assembler
Busybox	embedded	C
GNU	general software applications	C
Perl Power Tools	portability to any Perl platform	Perl

For our tests, we chose the four programs available in each of these collections: *cat*, *wc*, *md5sum*, and *grep*. We ran tests on two versions of each implementation. Table 2 shows the lines of code for the 31 programs under test. (The Perl Power Tools collection did not include *md5sum*, and so a Perl implementation of that program was from the CSPAN collection, and thus did not change between versions of PPT.) Program size increased between versions for most of the implementations. In 2 cases, program size decreased slightly. The GNU versions of *grep* were much larger than the corresponding versions in other collections, at least in part because GNU implements far more options for *grep*. Table 3 shows the number of options supported for all the programs. The number of option supported was determined during the testing experiments.

Table 2: Lines of Code in the programs being tested.

	asmutil		**GNU**		**Busybox**		**PPT**	
Program	0.14	0.17	grep 2.3, textutils 2.0	grep 2.5, textutils 2.1	0.50.3	0.60.4	ppt-1	ppt-0.12
cat	72	70	823	839	53	53	173	185
wc	255	264	371	371	156	169	342	342
md5sum	550	550	635	635	957	1074	624	
grep	167	305	16045	23141	289	372	621	612
mean	261	297	4468	6246	363	417	440	440
	279		5357		390		440	

Table 3: The Number of options supported

	asmutil		**GNU**		**Busybox**		**PPT**	
Program	0.14	0.17	grep 2.3, textutils 2.0	grep 2.5, textutils 2.1	0.50.3	0.60.4	1	0.12
cat	0	0	10	10	0	0	7	7
wc	3	3	4	5	4	4	6	6
md5sum	0	0	4	4	6	6	0	
grep	1	4	35	45	8	10	21	21
mean	1	1.75	13.25	16	4.5	5	8.75	8.75
	1.37		14.6		4.75		8.75	

Tests were run with 21 different input files as input. The input files used were called zerolength (an empty file), onechar (containing only the character “m”), , paper1, paper2, paper3, paper4, paper5, paper6, (all 6 are ASCI text files), 50lines (contained 50 lines of paper1),obj1, obj2, (executable files) progc, progl, progp, (source code in the languages

C, Lisp, and Pascal) trans (a transcript of a terminal session containing terminal control characters), geo (GIS data), bib (bibliographic data), news (an archive of USENET news messages), pic (image data), book1, and book2 (two full length books). Most of these files came from the Canterbury text compression corpus available at http://corpus.canterbury.ac.za/.

Tests were run on a Pentium-class PC running Redhat Linux 6.1. All one and two option combinations of command-line options were tested with all the 21 input files. Our tests produced 30,434 output files requiring about 1.7 gigabyes of disk space. Using N-Version techniques, we were successful at eliminating almost all of these files automatically, allowing us to focus on a few hundred files that had to be examined to determine if substantial disagreements (not just formatting discrepancies) existed between the programs tested.

Functional Agreement Among the Collections and Between Versions

Table 3 illustrates a fundamental problem in comparing the different implementations, and an immediate indication of the diversity among UNIX Utilities: there is no uniformity in the number of options supported. The *grep* program is a dramatic example. On the one hand, GNU supports 35 options in its first version, and 45 in its second version; on the other hand, the Assembler collection only supports 1 option in its first version and 4 in its second version. This kind of variation is a dramatic illustration that UNIX Utilities have not completely converged, and are unlikely to do so in the near future.

However, when the implementations support a particular functionality, they often agree to a high degree. For example, for a "basic" invocation of *cat* (using printable text only and no options), all the collections agreed in both versions. In general, when disagreements occurred, they were discovered during the runs that used the input files zerolength, obj1, obj2, progc, progl, progp, or trans.

In all cases, the two versions from the same source agreed with each other during our tests. Also, if two different implementations of the same program differed between collections in the first version, the differences persisted in the second versions. Each of the next paragraphs discussed differences discovered for each of the programs tested.

The program *cat* has an option *–n* that's implemented in the GNU and Perl versions. The intent is to number all the lines in the input file. The Perl version and the GNU version handle the % sign differently. In the Perl version, % is processed one way by the Perl interpreter, and in a contrary way by the *cat* application program, leading to somewhat confusing results. The option *–b* is again supported by GNU and Perl, and again there are disagreements. This option is supposed to number non-blank lines. In addition to the % symbol problem, the implementations differ on whether a line containing a single tab character should be counted as non-blank. The combination of the two options, *-nb*, is handled differently by GNU and Perl. GNU marks only non-blank lines, and Perl marks all lines. The *–e* option for *cat* is to end each line with a $ sign. GNU and Perl usually agree, but they disagreed on the *trans* file, which ends with repeated null characters. Perl treats the repeated null characters as an additional line, and GNU doesn't. Finally, the *–v* option is supposed to show non-printing characters using Caret and M- notation. GNU and Perl disagreed about how to handle particular characters such as tabs.

The program *wc* counts the number of lines, words, and characters in a file or files. The Perl implementation accept some of the options without changing its result. As for the actual counting, the GNU and Busybox versions always agreed on the number of words, but the Perl version often (though not always) reported a higher number of words. The assembler version often reported a smaller number of words. For textfiles, the difference in the counts was within 5%, but the differences were larger for files with non-printable characters. The implementations had several disagreements when input the file containing the single character "m." Although they all agreed there was a single character, The Perl implementation had an overflow error in the line count, reporting 4,294,967,295 words; Busybox and GNU reported no lines and one word; while the assembler reported no lines and no words.

The program *md5sum* calculates checksums for files and data streams. The tested programs do not implement any of the same options, but they agreed on all inputs when used without options.

The program *grep* is the largest that we tested. *grep* searches for occurrences of string patterns in files. In our experiments, we used the simple pattern "a" for all our searches. GNU and Perl implementations do not agree on several options. For example, the *–C* option is interpreted by the Perl implementation as a command to count the number of matches in a file, whereas the GNU version displays two lines of context around each match. (The assembler and Busybox implementations do not support any options for *grep*.) The GNU version did not search non-text files, terminating with an error message, whereas Perl searched the non-text files. (The GNU version was perhaps trying to avoid sending control characters to a screen, as this might disrupt a terminal session.) The Perl implementation supported a *–R* recursion option that wasn't supported by any of the other implementations. This option resulted in a system crash during our automated testing, and wasn't investigated further.

Differences in Execution Times

Despite the exceptions noted above, we were impressed by the general agreement on most of the tests. There were, however, dramatic differences in the time required to run the tests. Tables 4 shows those differences. The time taken by the assembler code was especially surprising, since assembler is often used to speed up implementations.

Subsequent examination of the assembler code for *grep* revealed that it fetched characters from the file one character at a time, resulting in many more system calls and delays than encountered by the other implementations. We had also expected execution times for Perl that were more competitive with GNU's C implementations, since many of Perl's subroutines are implemented as C code. We suspect that the simplicity of our searches (we always searched for the single character "a") may have biased our results somewhat. Still, the GNU execution times are impressive.

Table 4. Time taken to execute the tests. Time was measured using a high resolution timer with nano-second accuracy. Times listed are in milliseconds.

	Assembler		**GNU**		**Busybox**		**Perl**	
Program	0.14	0.17	grep 2.3, textutils 2.0	grep 2.5, textutils 2.1	0.50.3	0.60.4	ppt-1	ppt-0.12
cat	5.4	5.6	6.4	6.5	109.1	6.2	107.1	107.1
wc	13.6	14.2	13.8	15.1	62.6	61.9	331.3	330.5
md5sum	18.1	18.3	13.0	12.3	10.6	18.0	27.0	
grep	609.2	619.2	8.3	8.4	79.0	78.7	113.2	113.2
mean	161.5	164.3	10.3	10.5	65.3	41.2	144.6	144.4
	162.9		10.4		53.2		144.5	

Conclusions

OSS proponents think that over time UNIX utilities are likely to converge to the same ("correct") behavior; OSS opponents think that they are likely to diverge instead.

Like so much in the messy world of empirical software engineering research, our results were mixed. The implementations agreed on much of the functionality, especially when options were not invoked. This agreement seems particularly notable since the different collections do not share any centralized, formal control or documentation. Thus, among different collections, there was some evidence that some convergence has occurred, although no further convergence among the collections was evident between versions. Our tests did show functional stability *within* each collection between the versions for options in both versions.

However, there is wide variation in the number of options supported by the four implementations. When the implementations supported common options, there were some significant differences in the way files containing non-printable characters were handled. So convergence has not be complete.

The range in performance statistics was striking. For example, for *cat*, the performance measures ranged from a best of 5.4 milliseconds to a worst of 109.1 milliseconds. On an optimistic note, the worst performer in its first version improved its performance to 6.2 milliseconds in the second version tested. There were also wide variations in the amount of code required in the four different implementations. For example, the lines of code to implement *grep* varied from a low of 167 to a high of 23141 lines.

The N-version testing techniques seemed to be an effective method for exploring these implementations. In our opinion, our testing results give a far more detailed view of the actual behavior of these collections than do the documentation available from the collections themselves, or from external documentation about UNIX utilities.

References

Cingely, R. (2003). Unplugged: How Microsoft's misunderstanding of Open Source hurts us all. (October 23, 2003). http://www.pbs.org/cringely/pulpit/pulpit20031023.html, retrieved December 10, 2003.

Grissom, S. and Miller, K. (1999). N version testing in the undergraduate curriculum. *Computer Science Education*, Vol. 9, No. 1, 1-7.

Knight, J. and Leveson, N. (1986). An experimental evaluation of the assumption of independence in multi-version programming. *IEEE Trans. on Software Engineering*, Vol. SE-12, No. 1, 96-109.

Martin, M. (2003). New findings shake up Open-Source debate. *NewsFactor Network* (September 18, 2003). http://sci.newsfactor.com/perl/story/22319.html, retrieved December 10, 2003.

McKendrick, J. (2003) Ballmer: Open Source is not trustworthy. *ENT News* (October 22, 2003). http://www.entmag.com/news/article.asp?EditorialsID=6004, retrieved December 10, 2003.

Miller, B., Fredriksen, L. and So, B. (1990). An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, Vol. 33, No. 12, 32-45.

Miller, B., Koski,D., Lee, C., Maganty, V., Murthy, R., Natarajan, A. and Steidl, D. (1995). Fuzz revisited: a re-examination of the reliability of UNIX utilities and services. Technical Report CS-TR-1995-1268, Univ. of WI, Madison.