Analogues: Simplified Version Management

Jon Boone
Drexel University
jab545@drexel.edu
6/08/16

ABSTRACT
Version management is a black-art. Developers (and the occasional power-user) use little-understood, and extremely complicated Version Control Systems (VCSs) to manage their important artifacts. Meanwhile, other users are left behind, resorting to ad-hoc versioning or, worse, no versioning at all.

I envision a simplified interface to one of the most popular VCSs (git) which will enable the normal user to begin using systematic versioning with their artifacts. By presenting the user with the most important versioning capabilities via an extremely simple user interface, the goal of enabling users to adopt systematic versioning is achieved.

Author Keywords
Version Control; Minimal UI.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous. D.2.7 Distribution, Maintenance, and Enhancement: Version Control.

INTRODUCTION
Computing environments have progressed significantly since the early days of computing. Yet, one thing that has been lost in the progress is a simple way of maintaining multiple versions of a document. Forty to fifty years ago, operating systems (of which TENEX[1] for the DEC PDP-10 and TOPS-20[8] for the DECSYSTEM-20 are two examples), included versioning filesystems which automatically provided this functionality to some degree. As modern operating systems have evolved toward compatibility with UNIX™ and subsequently the POSIX standard, this feature has largely been dropped, although there are efforts to revive it such as PeriFS[11].

The result is that file versioning has become the province of programmers and the occasional power-user. As the target audience of the feature has shifted, so too has the feature set. Indeed, modern Version Control Systems (VCSs) provide so many features that it is difficult to consider a simple numbering scheme to track different versions of a file as a legitimate attempt at providing version control.

Analogues provides a simplified user interface to a popular VCS called git[4], exposing only the most commonly needed features, while maintaining compatibility with git itself. Like the filesystem versioning systems of old, the user of Analogues is shielded from the conceptual baggage that accompanies using such a powerful tool. The simple user interface enables a simple conceptual framework related to file versioning, not too dissimilar to that which is frequently used by non-power-users today.

RELATED WORK
Many papers have been published on VCS systems, though most of these tend to be focused on the programmer-as-user community. Since Analogues is not meant for programmers, many of the papers on version control are not directly relevant.

One particularly interesting paper by Coakley[2] introduces the idea of revision control to Microsoft Word documents. While the authors go to great lengths to make revision control accessible to the same user community that Analogues targets, their approach is to provide a similar mechanism as provided by git to the the target user community, coupled with a somewhat easier interface. Unfortunately, the approach does little to nothing to hide the more complex concepts of version control such as branching and merging. In contrast, Analogues attempts to remove the necessity of exposing these concepts to the user by not providing the branching and merging features.

There is some additional system work that has been done recently to add some features of version control back into modern operating systems for use by non-programmers, as well as enabling programmers themselves to use git in a more user-friendly manner. I choose to highlight two works in particular due to the similarity with Analogues.

Versions (Mac OS X 10.7+)
With the introduction of Mac OS X 10.7, colloquially known as OS X Lion, Apple introduced a versioning system called Versions[10]. With an interface similar to their Time Machine program for backup management, Versions allowed the user to visualize prior versions of a file that were available. This visual metaphor of a time-line, familiar to so

many people, shows that Versions was targeted at a similar user-base as Analogues.

Unlike Analogues, however, there did not appear to be a convenient way to store metadata related to each specific version that was readily available to the user. Similarly, unlike a true versioning filesystem, in Versions it was necessary for the user to specifically select to "Save a Version" – a feat which could only be accomplished if the application developer had integrated the proper framework to support this feature. Apparently, this was an uncommon choice for application developers, as the feature appears to have been removed from subsequent releases of Mac OS X. The only remaining documentation on it appears to be a support knowledgebase document on Apple's website.

Gitless

Another relevant work is an open-source program called Gitless. A self-described *experimental* VCS[5], it attempts to lower the conceptual burden on programmers, while maintaining the majority of features that programmers need. Started in July of 2013, with pre-release 0.0, it has progressed to pre-release 0.8.2 as of September of 2015[6]. Even with the reduced conceptual burden provided by Gitless, the sheer number of features supported makes learning this system daunting for a non-power-user (and perhaps a number of power-users as well).

Despite the conceptual similarity and approach of Gitless to Analogues, the target user-base is dissimilar enough from that of Analogues to warrant Gitless as not suitable for the non-power-user. Gitless is primarily a command-line interface "porcelain" over git, while Analogues is a GUI-focused application which makes use of the common desktop filesystem interface metaphor.

ANALOGUES: SYSTEM IMPLEMENTATION

System Design

Analogues was developed in phases. Initially I created a design specification, which included identifying the target audience and laying out the conceptual, semantic level and syntactic level models. It was during this phase that the key goal of simplification of version management was identified and refined.

Of particular importance was keeping the conceptual model as simple as possible. This meant sacrificing powerful features of common VCSs in order to stay true to the primary goal. Two important and powerful features that were identified as out of scope for the approach were branching and merging of branches, as these are frequently used – and just as frequently poorly understood – by many programmers. Resolving merge conflicts, which must be done by hand in many cases, is something that even sophisticated developers find challenging; to force this upon the target user base of Analogues would be inappropriate.

Figure 1: Initial Startup Screen - Low-Fidelity Prototype

The next phase involved creation of a low-fidelity prototype and the heuristic evaluation of that prototype. Using a prototyping tool called Balsamiq, I was able to create the prototype which allowed me to indicate what user interface elements would be present, as well as to provide a structured "walk-through" of how Analogues would potentially be used by a first-time user. The subsequent heuristic evaluation of the prototype further cemented my commitment to keeping things simple – from which concepts to expose, all the way to how to enable the execution of actions.

Figure 1 illustrates that the low-fidelity prototype's initial launch screen is clear of unnecessary clutter and provides only the minimum necessary actions to the user on the easily anticipated "File" menu. Figure 2 illustrates that the high-fidelity prototype's initial launch screen is similar, but establishes two visual fields of reference which are consistently presented throughout the use of Analogues.

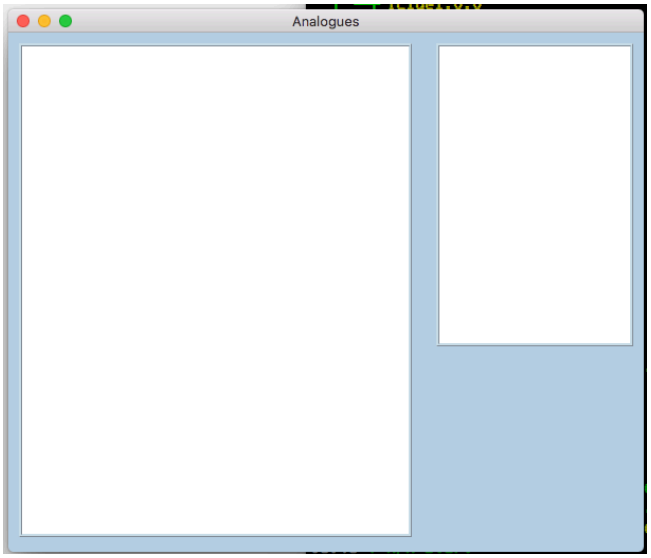

Figure 2: Initial Startup Screen - High-Fidelity Prototype

Another key decision point informed by this phase was the choice of implementation language(s) and targeted platforms. Because I feel strongly that version control for non-power-users is important, I wanted to have the versatility to support Windows, Mac OS X and Linux, to whatever extent possible. Choosing Java as the implementation language would have provided a consistent user-interface across all of the targeted platforms, but at the cost of not truly appearing native on any of them.

However, by choosing web technologies and the right frameworks, it became possible to readily support two of the three platforms with native look and feel on both. Correspondingly, Analogues is written in Javascript/HTML/CSS. By virtue of being developed using these technologies, the same code can run on all three of the target platforms. The sole pre-development dependencies are version 6.2.0 of node.js, which includes the corresponding version 3.8.9 of npm (the package management system for node.js) and a text editor.

To achieve the desired native look and feel, however, it was necessary to leverage an application framework called Electron. Electron is an open-source project that is maintained by GitHub that was originally used to create the open-source editor Atom[3]. It uses the Chromium open-source browser implementation to provide native look-and-feel windowing capabilities, while being completely controllable by Javascript[3]. This is conceptually the same as targeting one browser that runs on all of the targeted platforms and leveraging standard web technologies.

Most of Analogues runs in front-end Javascripts, including the interactions with the local filesystem for version repository maintenance. On the back-end, native menus and application control is provided by a node.js process that runs for the life of the application. Electron provides limited direct manipulation of the front-end by the back-end and vice versa, but for complicated interactions the recommended approach is to open a socket between the two and communicate over a custom protocol.

Git compatible repositories that store versioning information are made possible by the use of the libgit2 Node.js language bindings called node-git[9]. Created by GitHub, libgit2 is capable of being leveraged in nearly any programming language via provided language bindings – or as a Foreign Function Interface (FFI) to the library, which is written in C[7].

Analogues makes use of an application specific directory hierarchy ($HOME/.analogues) in which it stores the repositories of the files which are being managed. When first run, Analogues creates the repository directory hierarchy root and configures the minimal necessary git configuration values (user.name and user.email), if they are not already specified in the user's $HOME/.gitconfig file.

To add a file to versioning, Analogues creates a repository under $HOME/.analogues that ends in the name of the file. Inside this repository are stored various git-related files, as well as a file that stores the original path to the versioned file. At the time of this writing, directory hierarchies are not supported in Analogues, so each file added must have a unique name when the directory hierarchy has been stripped off.

Use of the System

Analogues allows the user to perform only five actions related to versioning: 1) add a file to versioning, 2) increase the version number of the selected file, 3) substitute a prior version of the selected file for the current version, 4) rename the selected file and 5) remove the selected file from versioning. At the time of this writing only 1), 2), and 5) are implemented.

The most fundamental use of Analogues involves adding a file to versioning, which can be accomplished by selecting the appropriate option from the "File" menu – or its accompanying keyboard accelerator. Figure 3 illustrates how the user is presented with a traditional "file navigation" dialog box which allows them to select one file to add.

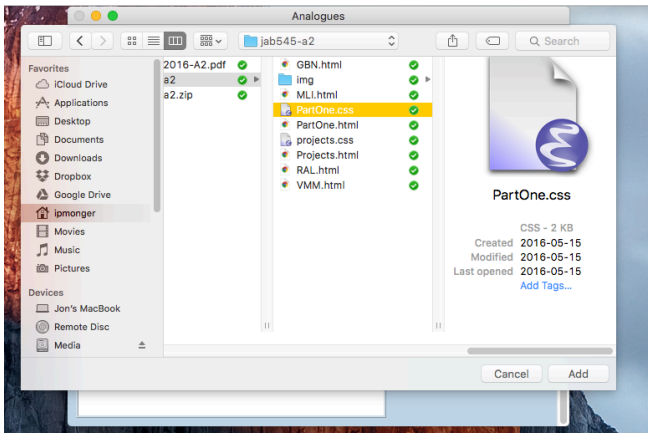

Figure 3: Selecting a File to Add

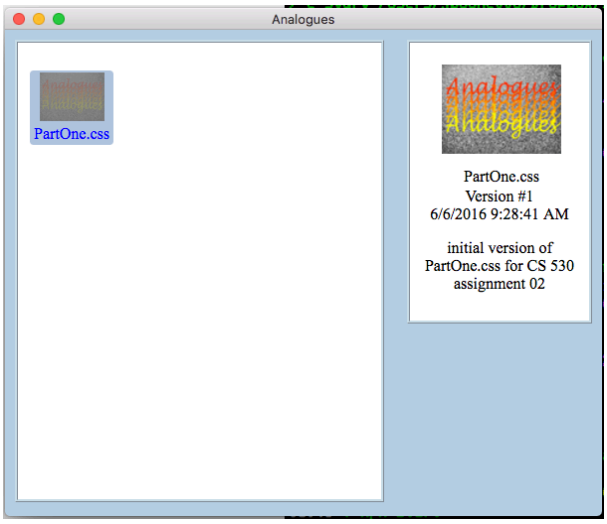

Figure 4: PartOne.css Added

Figure 4 is a picture from the prototype that shows the result of adding one file --- PartOne.css – into the repository. The detail-view on the right include the version number as well as the version comment.

To demonstrate the git compatibility, Figure 5 is a picture demonstrating the use of the command-line git utility to examine the resulting repository from adding "PartOne.css" to the repository. It shows that the initial repository commit has been successfully created and meta-data in the form of the version message has been stored along with it to document unique features of this version of the file.

Figure 6 shows the alert displayed to the user when a selected file is chosen for removal from versioning, which completely removes the version history as well as the file itself from the application specific directory hierarchy.

```
09:44 $ git log
commit 417223a02b092d95ae2ec0d46ae4bed38f79f8ac
Author: Jon Boone <jon_boone@cable.comcast.com>
Date:   Mon Jun 6 09:28:41 2016 -0400

    initial version of PartOne.css for CS 530 assignment 02
"/.analogues/PartOne.css [master Li
09:44 $
```

Figure 5: Git command line usage

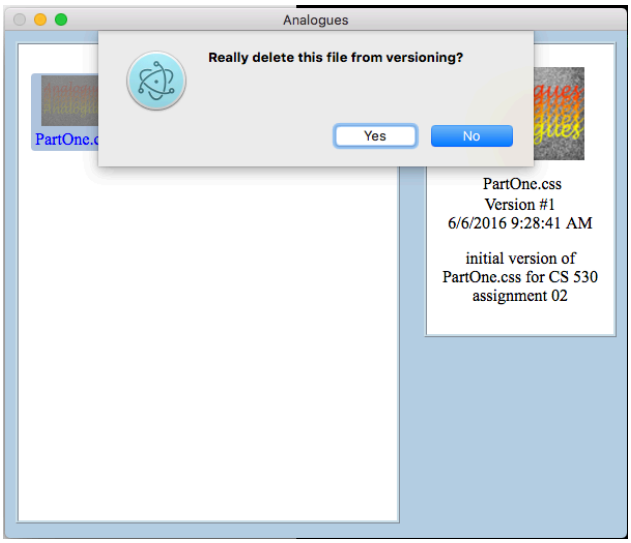

Figure 6: File removal alert dialog

EVALUATION

The targeted user-base for Analogues typically either does nothing to save differing versions of the files they create, or uses a system reminiscent of that provided by a versioning file system – the document name has the current version embedded into it. Analogues supports versioning without this embedding the version number in the name of the file which allows file names to be more generally useful for the user, while also allowing for larger numbers of revisions than were typically provided by versioning file systems, since no portion of the filename is reserved for the version number.

Analogues also provides one additional feature not typically available in a versioning file system, but which I anticipate will be well received by the target user-base – the addition of metadata support related to the rationale for the change. By providing a free-form text input to provide this rationale, users are encouraged to provide as much (or as little) information as they deem necessary to describe the important differences in the versions of their files.

Participants

In order to effectively evaluate Analogues, it is necessary to select evaluation participants from a range of ages and levels of computer sophistication. However, since Analogues is not targeted for computer programmers or power-users, it is equally important to ensure that the evaluation users do not fall into that category, out of a concern for bias. Power-users and programmers are more likely to be biased against Analogues as too simplistic and missing important powerful features, as well as being more likely to easily pick up on the concepts and workflows associated with Analogues.

Procedure

I chose three school-age children from 12-16 and one adult in the 40-60 age range as participants in the evaluation. None of them was a computer programmer or had any prior experience with a VCS. All were facile with basic computer operations using the standard desktop metaphors.

I provided them with a working prototype of Analogues, running on a Macbook, and a task set to accomplish. The task set involved creating revisions of a provided document and manipulating those revisions within Analogues. The provided document was a template document with Lorem Ipsum text.

In order to determine the ease with which an evaluation user is able to acclimate to using Analogues, a survey was provided via surveymoz.com.

Results

The core of the survey consisted of the following 6 questions, scaled on a range of 1-5 with 1 being most negative, 3 being neutral, and 5 being the most positive:

1. Did the user understand the purpose of Analogues?
2. Did the user find it difficult to make use of Analogues for the purpose as they understood it?
3. Were the options for manipulating files clearly indicated?
4. Were the menu items labeled in a clear manner?
5. Did the user feel that they understood how Analogues worked?
6. Would the user be willing to modify their workflow in order to accommodate the use of Analogues for important documents?

The results are captured in Table 1 (analysis courtesy of surveymoz.com).

Question	Mean	Variance	Std. Dev.	Std. Err.
1	3.75	0.19	0.43	0.22
2	3	1	1	0.5
3	2.75	0.69	0.83	0.41
4	4	0.5	0.71	0.35
5	3.5	0.75	0.87	0.43
6	4	0.5	0.71	0.35

Table 1: Survey Results

The results clearly indicate that, while the overall reception to Analogues was positive, there was more work necessary to make the program easy to use. In particular, observation of the evaluation users indicated that the implementation of the desktop metaphor was incomplete and that this caused some confusion on the part of the evaluators. Common options like right-clicking, drag-and-drop and selecting and deleting were attempted by multiple evaluators prior to choosing to utilize the menus on the task bar. The positive reception that evaluation users gave to the notion of using Analogues to version their important documents encourages further development of this project.

CONCLUSIONS

The Analogues prototype evaluation indicates that this is an excellent idea, although the current implementation is lacking in many respects. Chief among those is an explanation of what the purpose of Analogues is and a walk-through of how to use it. The use of node-git provides an "upgrade" path to users who later want to tackle the challenges of learning the git VCS.

Future Work

In the future, Analogues can be extended by improving the desktop metaphor implementation and providing a walk-through scenario upon startup. Labelling of menu items can also be improved to further clarify how to achieve the user's goal.

REFERENCES

1. Bobrow, D. G., Burchfiel, J. D., Murphy, D. L., and Tomlinson, R.S. *TENEX, A Paged Time Sharing System For The PDP-10*. BBN Report Number 2180. (1971).
2. Coakley, Stephen M., Mischka, Jacob, and Thao, Cheng. *Version-Aware Word Documents*. DChanges '14: Proc. 2nd Int. Wkshp. on Changes (2014).
3. Electron web site. http://electron.atom.io. Accessed: 2016-06-05.
4. Git web site. https://git-scm.com. Accessed: 2016-05-26.
5. Gitless: a version control system. http://gitless.com. Accessed: 2016-05-26.
6. Gitless source code repository on GitHub. https://github.com/sdg-mit/gitless/releases. Accessed: 2016-05-26.
7. Libgit2 web site. https://libgit2.github.com. Accessed: 2016-06-05.
8. Murphy, Dan. *Origins and Development of TOPS-20*. http://tenex.opost.com/hbook.html. Accessed: 2016-05-26.
9. NodeGit web site. http://www.nodegit.org. Accessed: 2016-06-05.
10. OS X Lion: About Auto Save and Versions. https://support.apple.com/en-us/HT202255. Accessed: 2016-05-26.
11. Ports, Dan R. K., Clements, Austin T., and Demaine, Erik D. *PersiFS: A Versioned File System with an Efficient Representation*. Proc. SOSP '05 (2005).