LINUX JOURNAL™
Since 1994: The Original Magazine of the Linux Community

HOW TO ADD SECURITY TO YOUR SSH KEYS

DECEMBER 2013 | ISSUE 236 | www.linuxjournal.com

BEST of LINUX

2013 READERS' CHOICE AWARDS

A LOOK AT USABILITY
IN OPEN-SOURCE SOFTWARE

INTERVIEW WITH AUSTRALIS
ARTIST, MUSICIAN AND LINUX USER

\+

CONSIDERING WOMEN AND LINUX

USING LVM
THE LOGICAL VOLUME MANAGER

WORKING WITH IMAGEMAGICK
ON THE COMMAND LINE

It's about the User: Applying Usability in Open-Source Software

You don't have to be an expert to apply usability tests in open-source software. Anyone can do it. And with good usability, everyone wins. JIM HALL

Open-source software developers have created an array of amazing programs that provide a great working environment with rich functionality. At work and at home, I routinely run Linux on my desktop, using Firefox and LibreOffice for most of my daily tasks. I prefer to run open-source software tools, and I think most *Linux Journal* readers do too. But as comfortable as the open-source software ecosystem can be, we've all shared or heard the same comments about some of our favorite Linux programs:

- "___ is a great program, once you figure out how to use it."
- "You can do a lot in ___, after you get past the awkward menus."
- "You'll like using ___, if you can learn the user interface."

That's the problem. No matter how powerful the program, that functionality is lost if people have to figure out how to use the program in order to unlock its secrets. Typical

users with average knowledge should be able to operate a general-purpose program. If a program is hard to use, that suggests the problem is with the program, not with the user.

Usability and Open-Source Software

"Usability" refers to how easily users can learn and start using software, or any similar "information product". Usability is separate from the functionality of the program, and so usability testing is different from unit testing. Instead, usability testing allows us to uncover issues that prevent users from using our programs.

Most open-source software programs are written by developers for other developers. Although some large open-source programs, such as GNOME and Drupal, have undergone usability testing, most projects lack the resources or interest to pursue a usability evaluation. As a result, open-source software programs often are utilitarian, focused on the functionality and features, with little attention paid to how people will use it. Applying usability practices tends to be antithetical to how open-source software is created. Open-source developers prefer functionality over appearance. Although some projects may have a maintainer who dictates a particular design aesthetic, many more do not. In an interview for this article, open-source advocate Eric Raymond commented to me that most programmers view menus and icons "like the frosting on a cake after you've baked it", which is an apt metaphor. Open-source software developers tend to prefer assembling the ingredients and baking the cake, not applying frosting to make it look nice.

So how can open-source developers easily apply usability to their own programs? There are many ways to implement usability practices. Alice Preston described 11 different techniques to evaluate usability in the STC Usability SIG newsletter. These methods run the gamut from interviews and focus groups to heuristic reviews and formal usability tests:

1. Interviews and observations: one-on-one sessions with users.

2. Focus groups: often used in marketing well before there is any kind of prototype or product to test, a facilitated meeting with multiple attendees from the target user audience.

3. Group review or walk-through: a

facilitator presents planned work flow to multiple attendees, who comment on it.

4. Heuristic review: using a predefined set of standards, a professional usability expert reviews someone else's product or design and shares comments with the designer.

5. Walk-around review: copies of the design or prototype are tacked to the walls of a conference room, and testers are invited to examine them and make comments.

6. Do-it-yourself walk-through: make mock-ups of the design, and use realistic scenarios to walk through the design yourself.

7. Paper prototype test: use realistic scenarios of a fake product that is still in design.

8. Prototype test: a step up from the paper prototype, test an animated mock-up against realistic scenarios.

9. Formal usability test: using a stable product, an animated prototype or even a paper prototype, test a reasonably large number of subjects against a controlled variety of scenarios.

10. Controlled experiment: a comparison of two products, with careful statistical balancing.

11. Questionnaires: ask testers to complete a formal questionnaire about how they would use a design.

However, such formal usability practices tend to clash with the open-source developer community and are like "swimming against a strong cultural headwind", to mix metaphors from Eric Raymond. With that in mind, developers should consider a subset of usability methods that apply well to the culture of open-source software. I propose this list:

1. Heuristic review.

2. Prototype test.

3. Formal usability test.

4. Questionnaires.

You don't need years of usability experience to apply good usability practices in open-source software development. As suggested by usability expert Janice (Ginny) Redish in numerous articles, you can learn a lot just by sitting down with

a few users and watching them use the software.

Whatever method you choose, the value of usability testing is in practicing it during development, not after the fact. Apply usability testing iteratively using prototypes, even paper-based mock-ups. At each round of testing, you will identify a number of issues that you can resolve for the next version. Successive usability tests will uncover additional issues and further improve your project.

Applying Usability Tests to Your Own Programs

Let's walk through a usability test as an example. Remember, the purpose of a usability test is to uncover issues that general users might have in utilizing the program. It is not a functional test of the program's features, but a practical test of its ease of operation, and as such, it differs from quality assurance or unit testing.

To start, define a set of written scenarios that represent how typical users with average knowledge would use the program. Don't look at every feature of the program, just describe the tasks that most users would want to do with the software. Make your scenarios realistic; provide a short description of each scenario, and ask the tester to perform tasks within that context. Use simple language; don't lead the tester by using the product's own words to describe menu items or actions, especially if typical users of average knowledge are unlikely to use those words every day. For example, if you want to evaluate an editor, your scenarios might ask the tester to type a short text document, save it and make basic copy edits to the file. For a Web browser, your scenarios could ask the user to search for a Web site, bookmark it and save a copy of the Web page for off-line use.

Invite testers to join you for a usability test. Although you might think you need a lot of users to evaluate a program's usability thoroughly, you really need only about five testers to get useful results, as usability expert Jakob Nielsen asserts in his research. Present the testers with the scenarios, one at a time, each on a separate piece of paper, and ask them to complete the tasks. Then simply observe what they do in the program, the routes they take to accomplish the tasks and the problems they encounter. Take plenty of notes.

The most difficult part of a usability test is watching a tester struggle to locate a menu or button. Although the correct action might seem apparent to you immediately, the

value lies in learning and identifying what is not obvious for other users. Do not give hints. If a tester is unable to finish a scenario, that's okay; just move on to the next scenario.

At the end of the scenarios, take a few minutes to ask follow-up questions of your tester. Identify any areas that seemed particularly difficult for the user. For example, you might ask "You struggled when you tried to do X; what would have made it easier?" or "What were you expecting to see on the screen when you were doing Y?" As a final wrap-up, ask the tester to describe what worked well in the program and what features should be improved.

Welcome to My Usability Test

I reviewed three common open-source projects in a formal usability test. I did this both to demonstrate the usability test process and to generate usability test results that could be generally applied to other open-source programs. Choosing the programs for my study required careful consideration. The ideal programs for my demonstration needed to balance multiple qualities: not be too big, because very complex menus can "lose" the audience in the details and confound the usability test results, and not be too small, as trivial programs will not support generally applicable conclusions. Further, the programs needed to be approachable by general users.

I solicited advice on several on-line forums, asking which open-source software programs had good usability. Sorting through the suggestions, three projects matched the criteria for my usability test:

1. Gedit (a text editor for GNOME).
2. Firefox (a popular Web browser).
3. Nautilus (a file manager for GNOME).

Because I work on a university campus, I invited students, faculty, staff and members of the public to participate in a usability study. I didn't ask for a specific level of technological expertise, as I was looking for typical users with average knowledge. In most formal usability tests, it's common to present each tester with a small gratuity; I gave out free pop and candy for them to take home with them after the test.

Although my preferred goal was about a dozen testers, I was satisfied with the seven who participated in the usability test. They ranged in age from about 20 to about 40, with three

men and four women. Most testers (five) claimed "low" experience with computers, and almost all (six) used Windows as their primary desktop. Testers used separate guest accounts on a laptop running Fedora 17 Desktop Edition, and so they started from the same initial default settings.

At the start of the usability test, I gave each tester a brief context of the usability study. I explained that this was a usability test, so it was about the software, not about them. If the tester experienced problems during the test, I let them know that would be okay, and we could move on to the next section. I was there only to observe their interaction with the software, not to judge their performance. Along the way, I said I would take notes and watch what was happening on their screens.

I also asked the testers to speak aloud what was going through their mind during the usability test. For example, if they were looking for a Print button, they should simply say, "I'm looking for a Print button." And, I encouraged them to track the mouse cursor on the screen with their eyes, so I could observe where they were looking for menus and buttons.

During the usability test, I presented the testers with a number of scenarios, each providing a brief context and an action they were to complete. For example, after asking testers to navigate to the BBC News Web site in the Firefox browser, one scenario asked them to increase the size of the text on the screen. It's important to note that the scenario did not use the same wording that was present in the menu action to apply the font size change:

> You don't have your glasses with you, so it's hard to read the text on the BBC News Web site. Please make the text bigger on the BBC News Web site.

Overall,the usability test included 19 scenarios, which testers completed in 30–40 minutes.

What Were the Usability Issues?

A heat map is a good way to represent the issues uncovered during a usability test. In Figure 1, each row represents the task, and each block represents a tester's experience. Green blocks indicate the tester was able to complete the task easily, usually on the first attempt. Orange and red blocks denote scenarios where the tester experienced difficulty or was unable to complete the task, respectively.

Interestingly, almost everyone

Gedit		G1. Type a sample note (provided) and save the file.
		G2. Edit text within the note.
		G3. Replace all instances of several words.
		G4. Save the file under a new name.
		G5. Change the default font.
		G6. Change the default colors.

Firefox		F1. Search for and navigate to the BBC News website.
		F2. Set the website as the default page.
		F3. Increase the font size.
		F4. Create a new tab and navigate to www.freedos.org.
		F5. Save a copy of a web page for offline use.
		F6. Download an image from the website.
		F7. Create a bookmark to the website.

Nautilus		N1. Create a folder.
		N2. Move the folder to a new location.
		N3. Rename the folder.
		N4. Create a bookmark or shortcut to a folder.
		N5. Delete a file.
		N6. Search for a file.

Figure 1. Usability Heat Map

experienced the same four issues:

- G5: change the default font in Gedit.
- G6: change the default colors in Gedit.
- N4: create a bookmark or shortcut to a folder in Nautilus.
- N6: search for a file in Nautilus.

In Gedit, testers were very confused about how to set the default font

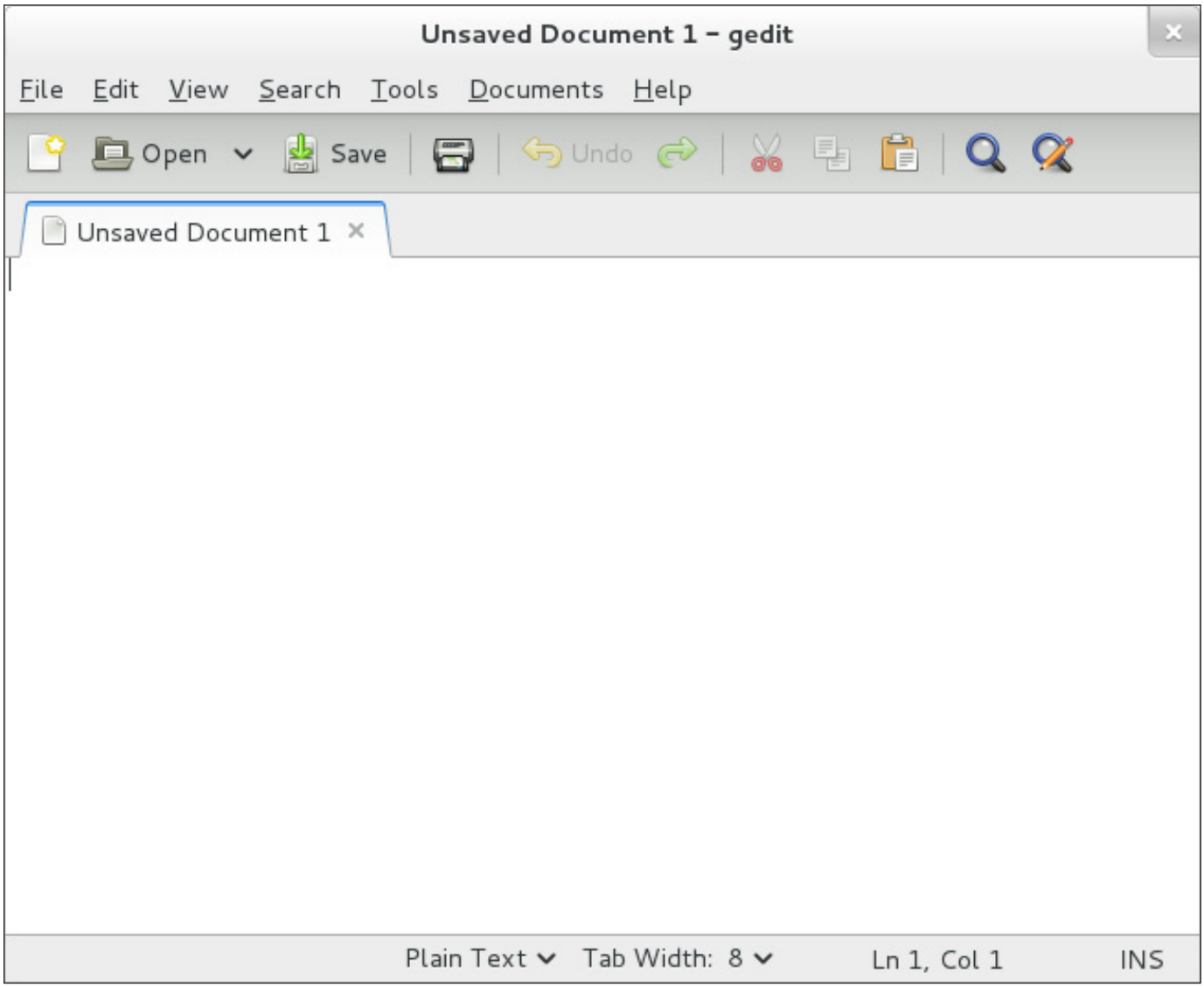

Figure 2. Gedit Screenshot—"How to Change the Font?"

and colors. Part of this confusion stemmed from thinking of the editor as if it were a word processor, such as Microsoft Word, which uses items on the toolbar to accomplish either action. Testers reported they were looking for a menu item called "Font". Failing that, testers also looked in File, Edit, View and Tools.

In Nautilus, testers became frustrated while trying to create a bookmark or shortcut to a folder, and the only user who successfully created the bookmark later commented she did so by accident. As explained in the scenario, the folder would be used for a project that collected photos and was located in the Pictures folder.

The most common action was to go into the Pictures folder, click on the

project folder, then select "Bookmarks - Add Bookmark". Nautilus doesn't display messages to the effect that "Add Bookmark" only creates a bookmark to the current location, not to a highlighted item, so testers were left confused when nothing happened.

Similarly, most testers found searching for a file in Nautilus a difficult task. They did not realize that the Search function starts from the current folder, not from their Home directory. Only two testers were able to use Search successfully. Of these, one happened to click on the Search button from the home directory. The other tried changing options in the drop-down Search action until eventually picking a combination that worked. One tester gave up with Search and

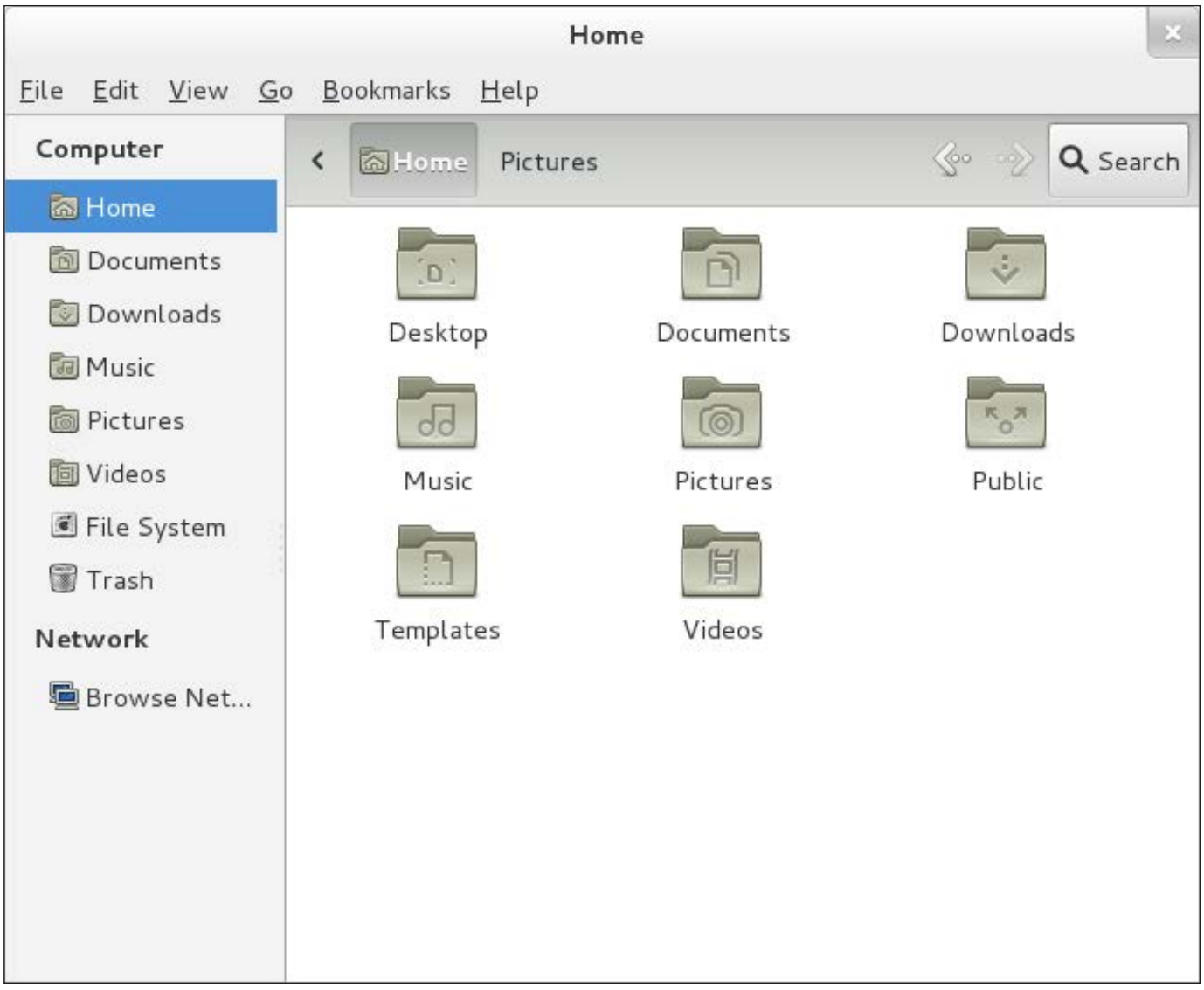

Figure 3. Nautilus Screenshot—"How to Create a Bookmark or Shortcut to a Folder?"

navigated into each folder in turn to find the file. Another user opted not to use Search at all, and used the same "seek and find" method.

And although GNOME was not part of the usability test, almost all testers experienced difficulty with the GNOME "Activities" menu hot corner. In the GNOME desktop environment, the Activities menu shows the list of available programs plus a view of the running applications. Users can bring up the Activities menu by clicking the menu icon in the upper-left corner of the desktop or by moving the mouse into that corner (the "hot corner"). Usually right away in the first scenario, testers would "overshoot" the program menu they were looking for, and hit the GNOME hot corner instead. This also occurred several other times throughout the usability test. Although testers were able to recover from the hot corner, it definitely caused frequent disruption.

What Worked Well for Usability?

Throughout the study, I observed four themes of good usability that allowed all testers to pass quickly through those parts of the usability test:

1. Familiarity: testers commented that the programs seemed to operate more or less like their counterparts in Windows or Mac OS X. For example, Gedit isn't very different from Windows Notepad or even Microsoft Word. Firefox looks like other Web browsers. Nautilus is quite similar to Windows Explorer or Mac OS X Finder. To some extent, these testers had been "trained" under Windows or Mac OS X, so having functionality (and paths to those features) that was approximately equivalent to the Windows or Mac OS X experience was an important part of their success.

2. Consistency: user interface consistency between the three programs worked strongly in favor of the testers and was a recurring theme for good usability. Right-click worked in all the programs to bring up a context-sensitive menu. Programs looked and acted the same, so testers didn't have to "re-learn" how to use the next program. Although the tool bars differed, all programs shared a familiar menu system that featured File, Edit, View and Help.

3. Menus: testers preferred to access the programs' functionality from the menus rather than via "hot keys" or icons on the toolbar. For example, the only toolbar icon that

testers used in the Gedit scenarios was the Save button. To complete other scenarios, testers used the drop-down menus, such as File, Edit, View and Help.

4. Obviousness: when an action produced a clear result, or clearly indicated success (such as saving a file in the editor, creating a folder in the file manager, opening a new tab in the Web browser), testers were able to move through the scenarios quickly. When an action did not produce obvious feedback, the testers tended to become confused. The contrast was evident when trying to create a bookmark or shortcut in the Nautilus file manager. In this case, Nautilus did not indicate whether the bookmark had been created, so testers were unsure if they had completed the activity successfully.

These are good lessons in open-source software and usability. Your program's user interface doesn't

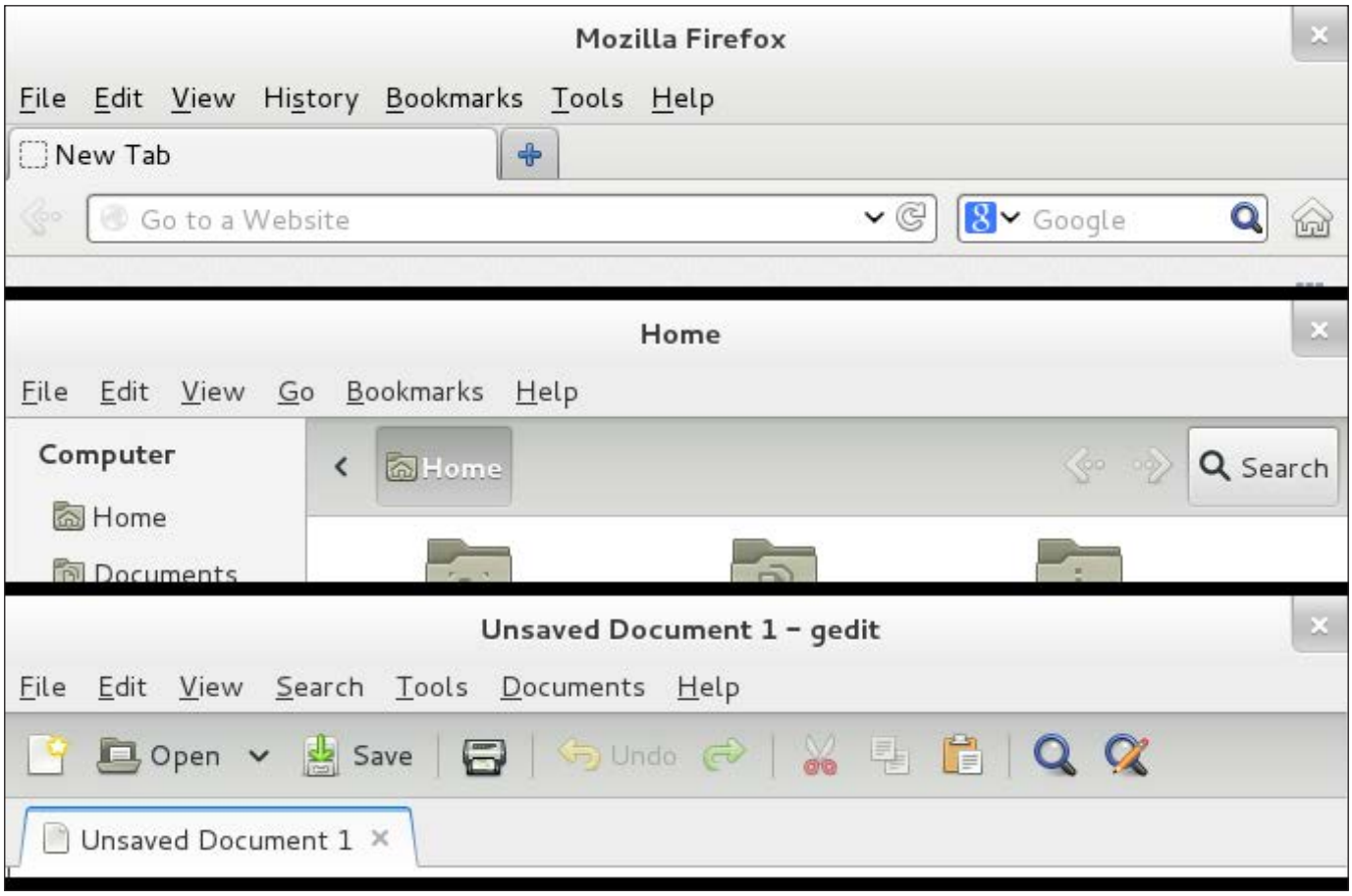

Figure 4. Firefox, Nautilus and Gedit Interface Comparison

have to be a beautiful impediment to understanding. Instead, leverage existing user interface paradigms. Be consistent with other programs on the same platform, whether they are other open-source software or proprietary programs. Use menus that are clearly labeled. Ensure that every action has a result that is obvious to the end user, especially if that result indicates a failure.

Where Do We Go from Here?

Usability should not be something that's just tacked onto a project or addressed only at the end of a development life cycle before releasing the next version of the software. Usability needs to be part of the design of open-source software, and addressed as part of a process. As open-source software developers, we generally are very good at applying good software development practices to our work. Now we need to take the next step and bring usability into that methodology.

Our next challenge in open-source software is finding ways to incorporate usability into our developer culture. This is a big step in open-source software. To date, usability has been antithetical to how open-source developers work. Most projects are written by developers for other developers. Crafting new functionality takes priority, and we rarely look at how our users will try to access those features.

In open-source software projects, the user community plays a strong part in testing each new release. Unfortunately, we cannot rely on the typical user-testing cycle to provide good usability feedback. Left on their own with no structure to usability testing, open-source software testers will respond with bland bug reports, such as "This feature is confusing." That's not helpful to a developer. In addition, usability researchers David Nichols and Michael Twidale comment in their published work that developers may not grant usability issues the same status as functionality bugs, leading to an inherent developer bias against usability bugs.

The approach to identify usability issues in open-source software, therefore, needs to be more structured. Open-source software developers can apply a variety of methods, although the ideal would be to conduct formal usability tests with a handful of users. Remember, you need only about five testers to get useful results.

Usability testing for open-source software projects doesn't need to be performed in a stuffy lab environment;

a project can find ways to "crowd source" usability testing with the user community. For example, the open-source Web content management system Drupal streamed testers' desktops as they undertook a usability test. This allowed Drupal developers all over the world to observe the usability test without having to travel to a single location. When developers can watch testers experience problems with their software, they better understand the issues and how to address them.

Another simple method is usability testing by "flash mob", a term suggested by Dana Chisnell on her Usability Testing Howto blog. To do testing via flash mobs, researchers simply can intercept people in a public space and ask them to try a few scenarios against a paper prototype and share their experiences. If each subject is willing to spare a few minutes, a "mob" of such testers will provide valuable feedback in a short amount of time. This idea of "flash mob" usability testing can be extended to other domains too. If your open-source software project is implemented as a Web site, you can conduct similar impromptu usability tests by intercepting Web visitors.

You don't have to be an expert to apply usability tests in open-source software. Anyone can do it. You only need to watch users try to use a program, and usability issues quickly will become clear. A handful of testers operating against a prototype can give you the feedback you need to make your open-source software even easier to use. And with good usability, everyone wins.■

Jim Hall is an advocate for free and open-source software, best known for his work on the FreeDOS Project. At work, Jim is the Director of Information Technology at the University of Minnesota Morris. He also is working toward his MS in Scientific and Technical Communication at the University of Minnesota.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

Resources

Open-Source Software and Usability (author's blog): **http://opensource-usability.blogspot.com**

More Information about Usability and Usability Testing: **http://usability.gov**

Janice (Ginny) Redish on Usability (good reference for Web developers): **http://www.redish.net**

Jakob Nielsen on Usability: **http://www.useit.com**