Olwen: The System Administrator

Characteristics

- Software Professional
- Uses Arch at work (don't ask)
- Most interested in reliability, stability, fine-grained control
- Needs to be able to automate tasks
- Needs to be able to receive alerts and view logs. Requires record keeping

Gwyn: The Power User

Characteristics

- Prefers doing work on the command line wherever possible.
- Prefers using the keyboard to control applications.
- Some understanding of the mechanics of a linux system.

Interests

- Works in IT, help-desk, linux support
- Programs casually.

Lleu: The Casual User

Characteristics

- Hasn't been using a Linux system for long.
- Primarily uses GUIs wherever possible.
- Not comfortable with the command line
- Prefers to use the mouse to control applications.
- Wants to spend as little time managing the installed applications as possible.

User Scenarios

Scenario 1: Lleu needs a pdf reader.

Lleu has been using their new Arch Linux system for a couple of weeks now; they've had some trouble, but they've managed to solve their problems mostly without help (other than documentation and tutorials) which has given them confidence. A colleague wants to show them a report they are working on, and sends Lleu a file over email. The file in the PDF format. When Lleu tries to read it they realise: their system doesn't have a PDF reader installed! If this were windows (which Lleu is familiar with) they would just go to a web browser, search for pdf reader, and then install the first option available (Adobe Reader). But since this is Linux, it's done differently, and Lleu knows this.

Lleu knows that to install software, you use a special program, the package manager, so they open up the package manager application. They're not sure where to go from here, but they know they want to search for something, so they look for something that looks like the search bars they are familiar with from web applications. Up the top of the window they notice what looks like a text-entry box with a little magnifying glass next to it, they guess this is it, and click on it; the box gets highlighted slightly, and this makes them confident that they application is listening for their input.

Lleu types in "pdf" into the box and presses enter, and a large number of results shows up. They scan through the results looking for something useful; the vast majority of the results are confusing to Lleu, talking about conversion, libraries, or other terms they don't understand, at least in this context. They find the first thing that seems to be a pdf viewer, open up it's information screen, and click install. The system seems to perform some function, eventually indicating that the program is installed. "Right!" thinks Lleu, "now I should be able to open the file!". They go back to the downloaded file in the web browser, and try and open it again. They get the same screen as before, asking them to choose an application to open the file with; this time Lleu notices that the program they just installed is in the list. They select it, and it opens. The reader app is a bit of a mess: a very old looking interface, with no clear controls or menus; thankfully it accepts mouse commands and Lleu can scroll through the document easily enough. They aren't particularly happy with the reader, and decide that when they have time they may try and look through the package list again to try and find a better one.

What do we learn from this scenario?

- Often, simple text search is not enough, and will give much too many irrelevant results. This is a very common situation in Linux package managers. Users could be helped in their search with the use of categories, recommendation or rating systems, or search hints (think like in google and other search engines).

- Having one central place to find applications to install can be tremendously useful, and will encourage users to go to that place when they are looking for a new program to fit some need. This can be clearly seen in Android and IOS through Google Play and the App Store respectively.
- Software varies massively in quality, especially in the open-source ecosystem. Often users will have no way of telling from the outset how mature, polished, or complete a particular application will be until they have already installed and run it; This can cause wasted time and frustration for the user. Again, a review/recommendation system could be particularly helpful here, as well as a more detailed description of the program's functionality.

Scenario 2: Gwyn is installing a C compiler.

Gwyn is starting a hobby programming project, and is trying to decide on what tools and languages to use to develop it in. She decides that she want's to give C another shot, because it would be useful to know for some of her other projects. However in the past she has had very bad experiences with the "standard" compiler, gcc. She decides to look online to see if there are any other compiler projects out there that are mature, modern, and which run on linux (she's aware of the Microsoft compiler system, but she isn't willing to switch back over to Windows to use it.

After some searching, she finds what she's looking for: the LLVM system, and the clang compiler that is a part of it. She opens up the package manager, and immediately searches for "clang" in the search bar; she's done this before and knows what she's doing. A number of results pop up, but since one is a complete match, it is put at the top of the list and is highlighted. She selects this option with her mouse, which annoys her somewhat, since she much prefers using keyboard shortcuts where possible, but doesn't know the keyboard shortcuts for this particular application, and doesn't know where to look to find them. In her experience they tend to be cumbersome or non-existent on GUI applications, so she hasn't bothered.

The info screen for clang pops up when she selects it. She selects install. Once it is done, she switches over to the terminal, writes a little "Hello World" application in C, and compiles it. It works smoothly, and she gets started with the project.

What do we learn from this scenario?

Scenario 3: Olwen needs to manage multiple versions of Java.

Olwen is managing their work development machine for a new Java project he has been assigned to. The project requires that it support multiple versions of java for machines that don't (or can't) have the latest versions; as a result, developers need to have multiple versions of Java installed simultaneously and test the software on each version.

Olwen starts up the system's package manager and does a search for "java". There are a *lot* of java related packages, and Olwen would find it very hard to find what he was looking for if the software didn't help out. The package manager understands that "java" is a core package that can have multiple different versions, so it has grouped them all together into a single "meta-package" and placed them at the top of the search results; Olwen clicks on this item and the system brings up the info screen. The screen shows all the available versions of the package, indicating which ones are installed. Olwen can select each version to toggle whether it will be on the system; he clicks on each of the versions that isn't yet installed, and then clicks the "install" button to confirm. The system brings up a window showing the changes, and asks for final confirmation.

Later on, management decided to drop support for a couple of the earlier versions of Java. Thankfully, the package system makes it easy for Olwen to just remove the versions he doesn't need.

TODO: FINISH

What do we learn from this scenario?

- Packages can have multiple versions that don't necessarily conflict. This can be seen in this example (with Java) and with other packages (like Python). Having a system which understands this, and allows the management of multiple versions can be very useful.
- TODO: MORE

Scenario 4: Lleu is trying to clear up disk space.

Lleu has run into a problem: their laptop has run out of disk-space (they knew they should have got the larger SSD), and they need to clear some space immediately. The package manager they're using shows the size of the files it's downloading, so Lleu guesses that there is probably a lot of space being taken up by the packages.

They open up the package manager, it opens to the standard main screen. For some reason, this system doesn't have the typical set of drop-down menus at the top of the screen like most programs they're familiar with (even many linux programs have them) and Lleu is feeling a little bit lost. But then they notice the bit "Help!" button over in the corner, in fact it is gently pulsating a bit to make itself seen; They click on it. A screen pops up with several bits of information: * An introduction to the program * A link to the full documentation * A series of keyboard shortcuts

Lleu opens up the full documentation and starts browsing. There are many sections, all dealing with some subsystem of the software. They find the section "Managing the package database" and look in there for something useful; after a couple of minutes of searching, they haven't found it. Back at the base of the documentation is a section "managing the package cache". "What's a cache?"

they think to themselves, they've heard the word in a non-computing context before, referring to some sort of stash of supplies; that sounds somewhat useful, so they read that section.

The documentation guides Lleu to a button on the left-hand side of the main screen, a little button with no text, shaped like a series of boxes; they mouse over the button and it comes up with a tooltip: "Manage Package Cache". Lleu clicks it, and a small section pops out from the left showing some information: the number and size of the packages stored in the package cache. At the bottom of this new section is a couple of buttons, the mouseover tooltips show "Conservatively clean cache" and "aggressively clean cache". They click on "aggressive" and a message pops up "Fully clearing the cache may prevent you from being able to downgrade packages in the future, are you sure?". Lleu thinks to themself: "why would it do that? shouldn't I just be able to download them again?" and clicks on "Yes". The cache clearing operation goes ahead, and clears up a significant amount of space. Despite the frustration with the interface and documentation earlier, Lleu is happy that the job is done, and they can continue using the computer.

What do we learn from this scenario?

- Important functionality should be easy to find as quickly as possible; a user shouldn't have to slog through documentation and confusing menus to find the single option they're trying to change. Drop-down menus *are* somewhat clunky, but they serve a purpose: putting away all those little options into sensible categories so that users can find them. While getting rid of drop-down menus seems tempting, make sure your interface can allow the users to access all the options through a more elegant method, or they'll just end up lost and frustrated.
- Documentation is hard to get right, but getting it right could be the most important thing you could do for usability; with excellent documentation, a harder system becomes much easier to learn.
- Good icons are also very hard to get right, trying to visually represent a complex action with a small picture can take a team of designers all on it's own. Good icons have to be clear, obvious as to what they represent, and evocative of the actual action. After the user figures out what icon corresponds to which task, memorization can do a lot of work.
- Language is important for usability. Novice users may not be familiar with the large amount of jargon terms that are commonly used with computer systems. These terms were made to be evocative and to form analogies to real-world objects or tasks, but the abstract nature of the terms (in the computing context) may not communicate what the designers want them to. Documenting terms can do some of the work, but often it is better to just not use jargon at all, and use more verbose or familiar terms. This all depends on your target users.

- The implementation details of computer systems can be unintuitive and confusing to users, both novice and advanced. TODO: Finish point

Scenario 5: Gwyn broke her computer with an update.

Gwyn is working on her software project, and has run into a problem. After stopping to have a break, turning off the computer, and then turning it back on when she returned, the OS has developed a problem. DESCRIPTION OF PROBLEM. Gwyn suspects that an update she did earlier in the day has broken something. With this possible cause in mind, she begins investigating by opening up the package manager. Up the top of the window are a number of icons representing the systems core functionality; one of these is an icon that appears to represent a clock. Guessing that this might be what she is looking for, Gwyn clicks on it. A new window opens up, and jackpot, it shows the history of changes made to the packages. Gwyn recognizes some of operations: the compiler she installed a week ago, a cleanup operation she did yesterday, and there it is: the update she did this morning.

When Gwyn clicks on the operation in the list, it opens up to show the individual packages that were updated as part of it. Additionally it has another drop-down menu showing the message log from the operation (what the system would have shown Gwyn when she performed the update)

TODO: Finish ### What do we learn from this scenario?

Scenario 6: Olwen is trying to replicate an install configuration on multiple devices.

What do we learn from this scenario?

Scenario 7: Lleu is trying to learn how to use the package managers advanced functions.

What do we learn from this scenario?

Scenario 8:

What do we learn from this scenario?