Software Release Practice HOWTO

Eric Steven Raymond

Thyrsus Enterprises (http://www.tuxedo.org/~esr/)

esr (at) thyrsus.com

도운이

권혁호

sobeit (at) nownuri.net

$Date: 2002/07/09 14:42:50 $

이 문서는 리눅스 오픈-소스 프로젝트의 올바른 공개 방법에 대해 설명한다. 여기서 설명하는 방법을 따른다면 사용자들이 당신의 프로그램을 설치하여 사용하고, 다른 개발자들이 당신의 코드를 이해하고 함께 개선하는 것을 가 능한한 쉽게 할 수 있다.

초보 개발자는 반드시 이 문서를 읽도록 하고, 숙련된 개발자도 새로운 프로 젝트를 내놓을 때 다시 살펴보도록 한다. 공개의 표준방법이 개선될 때마다 그것 은 주기적으로 수정되어 반영될 것이다.

고친 과정

고침 3.012 August 2000고친이 esr

First DocBook version. Advice on SourceForge and a major section on documentation practice added.

고침 1.01999/05/08　　고친이 임종호

최초 번역

This is version 3.0

1. 서론

1.1. 이 문서의 용도

오픈-소스 코드를 운영하고 사용하며 그 발전을 위해 협력하는 것을 돕는 표준방법이 존재한다. 이들 중 일부는 유닉스에서 사용되는 방법이거나 리눅스가 나오기 이전에 사용되던 방법이며, 다른 것들은 웹(WWW)과 같은 새로운 툴과 기술로 인해 근래에 발전한 것들 이다.

이 문서는 올바른 방법을 익히는데 도움이 될 것이다. 각 단락마다 점검항목 이 나열되어 있으며, 소프트웨어를 배포하기 전에 이에 따른 사전 점검을 하 도록 한다.

1.2. 이 문서의 새로운 버전

이 문서는 매월 comp.os.linux.answers (News:comp.os.linux.answers) 뉴스그룹에 게시될 것이다. 또, http://www.linuxdoc.org/LDP/HOWTO/Software-Release-Practice.html 을 통해서 최근 버전의 하우투(HOWTO) 문서를 볼 수 있다. 이에 대한 질문이나 비평은 자유롭게 Eric S. Raymond, `<esr@snark.thyrsus.com>` 앞으로 보내주기 바란다.

2. 프로젝트와 압축파일 이름 작성법

Metalab이나 PSA, CPAN과 같은 사이트를 유지하기위해 필요한 작업량이 증가함에 따라, (수작업이 아니라) 프로그램을 이용하여 관리작업의 일부 또는 전부를 처리하려는 경향이 늘어 나고 있다. 이러한 상황에서 컴퓨터가 이해하고 해석할 수 있는 규칙적인 형태로 프로젝트와 압축파일의 이름을 작성하는 것이 매우 중요하게 되었다.

2.1. 접두사-**major**번호.**minor**번호.**patch**번호 형태의 **GNU**식 이름 붙이기

압축 파일이 알파벳 소문자와 숫자로 이루어진 접두사(prefix), 이음선(dash) 그리고 버전 번호, 확장자(extension)와 다른 접미사로 되어 있는 GNU 형의 이름을 가지고 있다면 모든 이에게 도움이 될 것이다. version 1, release 2, level 3 인 ‘foobar’라는 프로젝트가 있다고 가정해보자. 만약 그것이 압축 파일의 한 부분(아마도 소스일 것이다)이라면 압축파일의 이름은 다음과 같을 것이다.

foobar-1.2.3.tar.gz

: The source archive

foobar.lsm

: LSM 파일(Metalab에 제출한다고 가정하자)

제발 다음과 같이 쓰지 마시오

foobar123.tar.gz
: 이 파일은 버전번호가 없는 'foobar123' 프로젝트의 압축파일로 여길 것이다.

foobar1.2.3.tar.gz
: 이 파일은 'foobar1'프로젝트의 버전 2.3인 압축파일로 여길 것이다.

foobar-v1.2.3.tar.gz
: 이 파일은 'foovar-v1'프로젝트로 여길 것이다.

foo_bar-1.2.3.tar.gz
: 밑줄은 읽고, 쓰고, 기억하기가 어렵다.

FooBar-1.2.3.tar.gz
: 당신이 바보(marketing weenie)처럼 보이고 싶은 것이 아닌 한, 이것 또한 말하고, 입력하고, 기억하기 힘들다.

소스와 바이너리 또는 다른 종류의 바이너리를 구분하거나 파일 이름에 제작 옵션을 표현하려면 다음과 같이, 버전번호 뒤에 오는 파일 확장자(extension)로 설명해라.

foobar-1.2.3.src.tar.gz
: 소스

foobar-1.2.3.bin.tar.gz
: 형식을 알수 없는 바이너리

foobar-1.2.3.bin.ELF.tar.gz
: ELF 바이너리

foobar-1.2.3.bin.ELF.static.tar.gz
: static link가 된 ELF 바이너리

foobar-1.2.3.bin.SPARC.tar.gz
: SPARC 바이너리

제발 'foobar-ELF-1.2.3.tar.gz'와 같이 사용하지 마라. 왜냐하면 프로그램은빵'-ELF'와 같은) 삽입사(infix)를 이해하지 못하기 때문이다. 일반적으로 바람직한 이름의 형태는 다음과 같은 순서로 되어 있다.

1. 프로젝트 접두사(prefix)
2. 이음선(-)

3. 버전 번호
4. 마침표(.)
5. "src" or "bin" (선택사항)
6. 마침표(.) 또는 이음선(-) (마침표를 선호한다)
7. 바이너리 형식과 옵션들 (선택사항)
8. 압축 확장자(extensions)

2.2. 적합하다면 지역적인 방법도 존중해라.

몇몇 프로젝트와 그룹들은 위에서 언급한 규칙과 호환성이 없으면서도 잘 정의된 이름과 버전번호를 가지고 있는 경우가 있다. 예를 들어, 일반적으로 아파치의 모듈들은 "mod_foo"와 같은 형태의 이름을 가진다. 그리고 자신의 버전번호와, 함께 사용되는 아파치의 버전번호를 모두 가진다. 유사하게, Perl 모듈은 버전번호로 소수점을 사용한다. (예. 1.3.3 대신에 1.303을 보게 될 것이다.) 그러므로 "Foo-Bar-1.303.tar.gz"는 1.303버전의 모듈 Foo::Bar를 의미한다.(그런데, Perl이 이러한 명명 법을 사용하기 시작한 것은 1999년부터이다.) 전문개발자나 전문가 그룹이 사용하는 관례는 존중해주어야 한다. 그러나 일반적으로는 아래의 가이드라인을 따르면 된다.

2.3. 입력하기 쉽고 고유한 접두사를 고르는데 신중해라.

접두사는 모든 프로젝트 파일이 공유해야하고, 읽고 입력하고 기억하기 쉬워야 한다. 그러므로 밑줄은 사용하지 마라. 그리고 아주 특별한 이유가 없다면 대문자로 시작하거나 중간에 대문자를 사용하지 마라. 이러한 것들은 눈으로 자연스럽게 읽는 것을 방해하고, 바보가 영리해 보이려는 것처럼 보인다. 접두사가 고유한 것이 아니라면 그것과 같은 이름의 접두사를 가진 프로젝트를 운영하는 사람들을 혼란스럽게 만들 것이다. 그래서 맨 처음 발표하기 전에 이름이 중복되는지 확인해 보라. 중복을 확인하기 좋은 두 곳은 index file of Metalab (http://metalab.unc.edu/pub/Linux)과 Freshmeat (http://www.freshmeat.net)(부록)이다. 또, 검사하기에 좋은 곳은 SourceForge (http://www.sourceforge.net)이다. 이들 사이트에서 검색을 해 보라.

3. 라이센스와 저작권 : 이론

라이센스는 주개발자와 동료 개발자들 그리고 사용자간에 이루어지는 사회적인 계약을 정의한다. 소프트웨어에 설정하는 저작권은 라이센스와 파생적으로 만들어지는 소프트웨어에 대한 권한을 법적으로 명기한다.

3.1. 오픈-소스와 저작권

공공소유권(public domain)에 해당되지 않는다면 대개 하나 이상의 저작권을 가진다. 베른(Berne)규약 (1978년부터 미국 법규에 적용된)에 따르면, 저작권은 외부에 명시할 필요가 없다. 즉, 저작자는 저작권에 대한 공지가 없어도 저작물에 대한 권한을 갖는다. 누가 저작자인가 하는 것은 복잡한 문제다. 특히, 여러 개발자의 공동작업으로 만들어진 소프트웨어의 경우에는 더하다. 이러한 경우 때문에 라이센스가 중요하다. 어떠한 자료가 사용되었는지를 문서에 설정함으로써 저작자가 임의의 행위를 하는 것으로부터 보호하면서 사용자에게 권한을 허용할 수 있다.

독점 소프트웨어는 라이센스가 저작권을 보호하도록 되어 있다. 이 경우 라이센스는 저작자에게 가능한 많은 법적인 테두리를 주는 반면 사용자에게는 적은 권한만을 허용하는 방안으로 사용된다. 즉, 저작자는 매우 중요하게되고, 라이센스 논리는 제한적이어서 거의 중요하지 않게 된다.

오픈-소스 소프트웨어는 상황이 정반대이다. 즉, 저작권은 라이센스를 보호하기 위하여 존재한다. 저작자가 가지는 단 하나의 권한은 라이센스를 집행하는 것이다. 그렇지 않으면, 극소수의 권한만이 보전되고 대부분의 선택은 사용자에게 귀속된다. 특히, 저작자는 당신이 이미 보유하고 있는 복사본에 대한 라이센스를 변경할 수 없다. 그러므로 오픈-소스 소프트웨어에서는 저작자보다는 라이센스가 훨씬 더 중요하다.

일반적으로 프로젝트의 저작자는 현재의 프로젝트 리더 또는 스폰서 조직이 된다. 새로운 프로젝트 리더가 나타나면 저작자도 종종 변할 수 있다. 그러나 이것은 그리 어렵지도 않고 심각하지도 않다. 많은 오픈-소스<65533>소프트웨어 프로젝트는 다수의 저작자를 보유하고 있다. 그러나 이로 인하여 법적인 문제가 발생한 예는 없었다. 어떤 프로젝트는 오픈-소스 소프트웨어를 유지하자는 주장에 따라 변호사에게 의뢰해 저작권을 자유 소프트웨어 재단(FSF)에 귀속시키기도 한다.

3.2. 오픈-소스 소프트웨어의 자격

라이센스는 목적상, 몇 가지 종류로 권한을 구분 한다. 복사와 재배포권한. 사용권한. 개인적인 사용을 위한 수정권한. 수정본 재배포권한. 라이센스는 이러한 권한을 적용하여 제한하거나 조건을 붙일 수 있다. Open Source Initiative (http://www.opensource.org)는 "오픈-소스" 또는 (이전의 용어로) "자유" 소프트웨어가 어떠해야 하는 지에 대한 많은 생각들의 결과로 만들어진 것이다. 이에 의해 라이센스에서 요구하는 사항은 다음과 같다.

1. 무제한의 복사권이 허용되어야 한다.
2. 무제한의 사용권이 허용되어야 한다.
3. 무제한의 개인 사용을 위한 수정이 허용되어야 한다.

오픈-소스 가이드라인은 수정된 바이너리의 재배포를 금지한다. 이것은 방해 없이 작업코드를 선적하기 원하는 소프트웨어 배포자들의 요구와 맞아떨어진다. 그리고 이는 저작자가, 수정된 소스의 배포는 원시코드에 패치본을 더하여 배포하도록 요구할 수 있게 한다. 그렇게 함으로써 저작자의 의도와 타인에 의한 어떠한 수정도 "감시 추적"할 수 있는 환경을 구성할 수 있다.

OSD는 'OSI가 인증하는 오픈-소스'의 법적인 정의이며, "자유 소프트웨어"처럼 누구나가 따를 수 있는 좋은 정의이다. 모든 라이센스(MIT, BSD, Aristic, GPL/LGPL)는 이를 만족한다.(단, GPL의 경우처럼 선택하기 전에 반드시 숙지해야 할 또 다른 제한이 존재하는 경우도 있다.) "GPL"이나 다른 표준 라이센스로 장식되어 있을지라도, 비상업적인 용도로만 사용하도록 되어 있는 라이센스는 오픈-소스의 자격을 얻지 못한다. 이런 라이센스는 특정 직업이나, 사용자, 그룹을 차별한다. 그리고 CD-ROM 배포자들이나 오픈-소스 소프트웨어를 상업적으로 확산하려는 사용자들을 골치아프게 만든다.

4. 라이센스와 저작권 : 실습

위에서 다룬 이론을 실제로 적용하는 방법을 알아보자.

4.1. 저작자를 본인 또는 자유 소프트웨어 재단(FSF)으로 설정해라.

만약, 변호사와 후원하는 조직이 있는 경우에는 저작권을 그 조직에 귀속시키기를 원할 수도 있다.

4.2. 오픈-소스 정의에 합당한 라이센스를 사용해라.

오픈-소스 정의는 라이센스를 위한 금쪽과도 같은 표준이다. OSD가 라이센스 그 자체는 아니다. OSD는 오픈-소스 라이센스로 간주될 수 있는 가장 기본적인 권한의 집합을 정의한 것이다. OSD와 기타 지원해야할 자료는 Open Source Initiative (http://www.opensource.org)에서 찾을 수 있다.

4.3. 가능하면 독자적인 라이센스는 쓰지마라.

널리 알려진 OSD-적합 라이센스는 잘 정리되고 해석된 전통을 가지고 있다. 개발자(확장해서 사용자)는 라이센스가 함축하는 바를 알고 있으며, 라이센스를 통하여 감당해야할 위험요소와 트레이드 오프 (trade-off)를 이해한다. 그러므로 가능하다면 OSI사이트에서 제공하는 표준 라이센스 중 하나를 사용해라. 꼭 독자적인 라이센스를 사용해야 한다면, 반드시 OSI의 인증을 득해야 한다. 그렇게 하지 않으면 많은 논쟁과 대가를 감수해야 한다. 이를 극복하지 못할 경우 얼마나 격렬한 라이센스 논쟁을 일으키게 될지 당신은 알수 없을 것이다. 사람들은 화를 낼 것이다. 왜냐하면 라이센스는 오픈-소스의 핵심 가치를 유지하는 신성한 서약과도 같은 것으로 간주되기 때문이다.

게다가, 당신의 라이센스가 법정에서 시험을 받는다면, 현존하는 문구 해석의 사례가 매우 중요하게 된다. 이 글을 쓰고 있을 때(2000년 중반)까지는 오픈-소스 라이센스와 관련해 부적절함으로 인한 소송사례는 없었다. 그러나, 법정에서 라이센스와 계약서를 그들이 속한 단체의 관습에 따라 해석하는 것은 (적어도 미국과 관습법을 적용하는 영국과 나머지 영연방 국가에서는) 일종의 사법적 교리로 통한다.

5. 개발 방법

이에 관한 내용들 중 대부분은 리눅스뿐만 아니라 다른 유닉스에서도 이식 가능하도록 프로그램을 작성하는 것과 관련이 있다. 다른 유닉스에 이식 가능하게 하는 것이 단지 전문가적인 훌륭한 형식이나 해커주의 때문은 아 니다. 그것은 리눅스 스스로 미래의 변화에 대비하기 위한 것으로 가치가 있다.

궁극적으로, 다른 사람이 당신의 코드를 리눅스가 아닌 곳에서 사용하려고 한다면, 이식성은 당신이 받게 될 성가시고 난처한 메일의 숫자를 최소한으로 줄여줄 것이다.

5.1. ANSI C나 이식가능한 스크립트 언어로 작성해라

이식성과 안정성을 위해 ANSI C나 이식 가능한 스크립트 언어로 작성해야 한다. 왜냐하면 다른 플랫폼에서의 실행을 위해서이다.

스크립트 언어로 적당한 것은 Python, Perl, Tcl, 그리고 Emacs, Lisp, PHP등 이다. 단순한 구식 shell은 적당하지 않다**.** 왜냐하면, 미묘한 특성에 따라 매우 다양한 구현상의 차이가 있을 뿐만 아니라, shell alias와 같은 사용자의 환경설정 변화에도 영향을 받기 때문이다.

자바는 이식성 있는 언어라고 믿어지지만 리눅스상에서 유용할 만큼 구현되지 않았고 리눅스 시스템과의 통합성도 부족하다. 자바의 성장으로 날로 인기가 높아지지만 자바는 여전히 힘든 선택이다.

5.2. C를 이식성있게 사용해라

C로 프로그램을 작성하는 경우 ANSI C의 모든 규정(모듈간의 불일치를 알수 있도록 도와주는 함수 프로타입을 포함한)을 사용하면 된다. 구식의 K&R 은 이미 한물 간 컴파일러이다.

반면에, GCC-specific 규정('-pipe' 옵션과 같은)또는 nested 함수가 적용 가능할 것이라고 추측하지는 마라**.** 이것들은 갑자기 나타나서 리눅스나 GCC를 사용하지 않는 시스템에 이식하려는 사람으로 하여금 당신을 괴롭히게 할것이다.

5.3. autoconf/automake/autoheader 사용해라

C를 사용해서 프로그램을 작성한다면, 이식성과 시스템 환경설정 그리고 Makefile의 적용을 위해<65533> autoconf/automake/autoheader를 사용해라. 요즘 소스를 이용하여 소프트웨어를 설치하려고 하는 사용자는 "configure; make"라고 치면 깨끗하게 프로그램이 만들어지기를 바란다. 그리고 그렇게 되어야 한다.

5.4. 공개하기전에 코드가 온전한지 검사해라

C로 프로그램을 작성한다면, 공개하기 전에 최소한 한번이라도 '-Wall' 옵션을 사용하여 컴파일 해보고 오류를 제거해야 한다. 이렇게 하면 매우 많은 오류를 발견할 수 있다. 철저하게

‘-pedantic’ 옵션을 사용해 컴파일하는 것도 좋은 방법이다.

Perl을 사용하였다면 공개 전에 ’perl -c’(해당된다면 ’perl -T’), 그리고 ’perl -w’를 사용해서 아주 세심하게 코드를 검사해야 한다.(Perl에 관한 문서를 참고해라.)

5.5. 공개하기전에 문서와 README 파일이 온전한지 검사해라

문서를 철자 검사기로 검사해라. 만약, 철자법도 모르는 것처럼 보인다면, 사람들은 당신의 코드도 부주의하고 변변치 않은 것으로 판단할 것이다.

6. 배포본 제작 방법

이 지침은 사용자가 배포본을 다운로드 받고, 검색하고, 압축을 풀때 배포본이 어떻게 보여져야 하는지에 대해 설명한다.

6.1. tar 파일은 항상 하나의 새로운 디렉토리에 풀어지도록 해라

초보 개발자가 범하는 가장 성가신 실수 중의 하나가 tar 파일을 배포본이 있는 현재의 디렉토리에 압축이 풀리도록 만드는 것이다. 이것은 현재 디렉토리에 이미 존재하는 파일을 덮어쓸 위험이 있다. 이런 실수를 절대로 하지 마라**!**

그 대신, 프로젝트의 이름을 따서 만든 하나의 공통 디렉토리를 포함하는 압축 파일을 만들어서, 이 파일들이 현재 디렉토리 아래에 위치한 새로운 디렉토리에 압축을 풀 수 있도록 해라.

여기 makefile 트릭(trick)이 있다. ‘foobar’라는 배포본의 디렉토리를 가지고 있으며 SRC가 배포본의 파일 리스트를 포함하고 있다고 가정하면 다음과 같이 하면 된다.

```
foobar-$(VERS).tar.gz:
@ls $(SRC) | sed s:^:foobar-$(VERS)/: >MANIFEST
@(cd ..; ln -s foobar foobar-$(VERS))
(cd ..; tar -czvf foobar/foobar-$(VERS).tar.gz `cat foobar/MANIFEST`)
@(cd ..; rm foobar-$(VERS))
```

6.2. README를 포함시켜라

`README` 또는 `READ.ME` 파일을 포함시키면 그것은 배포본의 지침서가 될 것이다. 오래된 관행에 따라, 소스의 압축을 푼 사용자는 이 파일을 가장 먼저 읽게 된다.

README 파일에 포함되어야 할 내용들은 다음과 같다.

1. 프로젝트의 간단한 설명

2. 프로젝트 웹사이트의 주소(만약 있을 경우)
3. 개발자의 제작 환경과 설치상의 잠재적인 문제점
4. 중요 파일과 하위 디렉토리 구성에 대한 설명(로드맵)
5. 제작/설치에 관한 설명이나 그것을 포함한 파일의 이름(일반적으로`INSTALL`).
6. 프로젝트 진행자/참여자의 명단이나 그것을 포함한 파일의 이름(일반적으로`CREDITS`).
7. 프로젝트의 최근 소식이나 그것을 포함한 파일의 이름(일반적으로`NEWS`).

6.3. 표준 명명(naming) 규칙을 존중하고 따르라

README 파일을 보기 전에도 용감한 탐험가(사용자)는 배포본의 최상위 디렉토리에 있는 파일을 훑어볼 것이다. 파일의 이름은 그 자체로 정보를 포함하고 있다. 일반적인 명명 규칙을 따름으로써 사용자에게 다음에 무엇을 보아야 할지 실마리를 제공해 줄 수 있다.

여기 몇 개의 일반적인 최상위 파일 이름과 그들이 의미하는 것이 있다. 모든 배포본이 이 파일들 전부를 필요로 하진 않는다.

README 혹은 READ.ME

가장 먼저 읽어야할 로드맵 파일

INSTALL

설정, 제작, 설치 방법

CREDITS

프로젝트 참여자 명단

NEWS

프로젝트의 최근 소식

HISTORY

프로젝트의 역사

COPYING

프로젝트의 라이센스(GNU 규정)

LICENSE

프로젝트의 라이센스

MANIFEST

배포본의 파일 리스트

FAQ

프로젝트에 관한 간단한 FAQ 문서

TAGS

Emacs나 vi를 위해 생성되는 tag 파일

대문자로 된 파일은 제작(build)을 위한 컴포넌트라기 보다는 패키지에 관한 정보(metainfomation) 를 포함하는, 사용자가 읽을 수 있는 파일임을 기억하기 바란다.

FAQ를 만들어 놓음으로써 상당부분의 고통을 덜 수 있다. 프로젝트에 관해 빈번하게 물어오는 사항들은 FAQ에 정리하도록 한다. 그러면 질문이나 버그리포트를 보내기 전에 FAQ를 읽어볼 것이다. 잘 정리된 FAQ는 사용자 지원에 대한 프로젝트 관리자의 부담을 엄청나게 줄여준다.

각각의 공개 배포본마다 시간정보를 포함하는 HISTORY 또는 NEWS 파일을 만드는 것이 좋다. 다른 어떤 점보다도, 만약 특허와 관련해서 소송이 발생했을 때 (아직까지는 그런 경우가 없지만, 있을 경우를 대비하는 것이 좋다.) 이것은 누가 먼저 시작했는지를 알려주는 주요 기록이 된다.

6.4. 업그레이드를 고려한 설계를 해라

새로운 공개판을 발표할 때마다 소프트웨어는 변하게 될 것이다. 이러한 변화 중에서 이전 버전과 호환이 안되는 경우도 있을 것이다. 따라서, 이런 경우엔 설치에 관한 디자인을 할 때 심각하게 고민해야 한다. 왜냐하면 똑같은 시스템에 여러가지 버전의 소프트웨어가 동시에 존재할 수 있도록 해야 하기 때문이다. 이것은 라이브러리에 있어서 특히 중요하다. API의 변화에 따라 모든 클라이언트 프로그램을 고정된 방식으로 업그레이드하는 것을 기대할 수는 없다.

Emacs, Python 그리고 Qt 프로젝트는 이를 처리하는 좋은 방법을 보여준다. 버전번호를 붙인 디렉토리 이름을 사용하는 것이다. 설치된 QT 라이브러리의 계층구조는 아래와 같다. (${ver} 은 버전 번호이다.):

```
/usr/lib/qt
/usr/lib/qt-${ver}
/usr/lib/qt-${ver}/bin          # moc의 위치
/usr/lib/qt-${ver}/lib          # .so의 위치
/usr/lib/qt-${ver}/include      # 헤더 파일의 위치
```

위와 같은 방식으로 여러 버전을 동시에 수용할 수 있다. 단, 클라이언트 프로그램은 사용하고자하는 라이브러리의 버전을 명기해야 하는 부담이 있긴 하지만, 이것은 인터페이스 자체를 완전히 바꾸는 것에 비하면 아주 조그만 부담에 지나지 않는다.

6.5. RPM으로 제공해라

설치할 수 있는 바이너리 패키지의 사실상의 표준 형식은 레드햇 패키지 매니저, RPM이다. 가장 인기 있는 리눅스 배포본에 사용되며 실질적으로 다른 모든 리눅스 배포본(데비안과 슬랙웨어는 제외; 데비안에서는 가능하다.)에서도 사용된다.

따라서, 프로젝트 사이트에서 설치 가능한 RPM과 소스 tar파일을 동시에 제공하는 것이 가장 바람직하다.

또 소스 tar 파일 내에 RPM의 스펙 파일을 포함시키고, makefile 안에 RPM을 생성할 수 있는 파일을 넣는 것이 좋다. 스펙 파일은 '.spec'이라는 확장자를 가져야 한다. 'rpm -t' 명령을 쓰면 tar 파일에 있는 스펙 파일을 찾을 수 있다.

Makefile과 version.h를 분석하여 자동으로 올바른 버전 번호를 설정하는 쉘 스크립트를 이용하여 spec 파일을 생성해라.

주의:소스 RPM을 제공한다면, 프로그램이 /tmp 또는 /var/tmp에 만들어지도록 BuildRoot를 사용해라. 그렇지 않으면, 'make install'과정 중에 설치 프로그램이 파일들을 실제 최종 위치에다 설치할 것이다. 이러한 일은 파일의 충돌이 있는 경우나, 패키지 설치를 원하지 않는 경우에도 일어날 수 있다. 이렇게 하면 모든 파일들은 설치되고 시스템의 RPM 데이터베이스는 이것을 알지 못한다. 이런 좋지 않은 SRPMS의 행위는 지뢰밭을 만들게되므로 삼가야 한다.

7. 문서화 방법

가장 좋은 문서화의 방법은 무엇인가를 써보는 것이다. 많은 프로그래머가 이를 빠뜨린다. 여기 문서화를 해야만 하는 두 가지 이유를 설명하겠다.

1. 문서화를 통해 설계 문서를 작성할 수 있다**.** 문서화의 가장 좋은 시기는 무엇을 해야할 지를 생각하는 동안이며, 한 줄의 코드도 작성하기 전이다. 프로그램이 어떻게 움직여야 할지를 자연어로 서술하다 보면, 프로그램이 무엇을, 어떻게 하도록 할 것인가와 같은 고차원적인 질문으로 생각이 발전할 것이다. 이렇게 함으로써 훗날에 있을지도 모르는 작업의 양을 줄일 수 있다.
2. 문서화는 코드의 품질에 대한 홍보이다**.** 많은 개발자들이 빈약하고, 불충분하며 문법적으로도 엉성한 문서를 만듦으로써, 사용자의 요구에 대해 무관심하고 무지하다는 인식을 심어주게 된다. 반면에 좋은 문서는 지적이고 프로의식이 있다는 인상을 주게 된다. 만약, 경쟁상대가 있는 프로젝트라면 최소한 잠재적인 사용자들이 프로그램을 제대로 보지도 않고 선택의 대상에서 제외하지 않도록 문서를 잘 만들어야 한다.

이 HOWTO는 실제적이지만 기술적인 작문 코스를 위한 것은 아니다. 여기서는 문서를 작성하고 표현하는데 필요한 형식이나 도구에 초점을 맞추어 설명한다.

유닉스와 오픈-소스 커뮤니티에게는 오랜 전통으로 내려오는 강력한 문서화 도구가 있음에도 불구하고, 여러가지의 다른 형식이 과다하게 존재하는 것은 문서화가 단편화되어 가는 경향이 있고, 사용자가 일관된 방법으로 브라우징하고 인덱싱 하는 것이 어렵다는 것을 의미한다. 여기서는 사용법과 강점 그리고 일반적인 문서화 양식의 약점을 요약하여 설명한다. 그리고 훌륭한 문서화 방식을 권고할 것이다.

7.1. 현재의 문서화 방법

여기서는 오픈-소스 개발자들 사이에서 널리 사용되는 문서화 표현 형태를 설명한다. ’presentation’은 (폰트 변경과 같은)문서의 외양을 명확하게 제어하는 마크업이며, ’structural’은 (절의 구분이나 강조 태그 같은)문서의 논리적인 구조를 설명하는 마크업이다. 그리고 ’indexing’은 사용자가 전체 문서 집합에서 관심있는 자료를 확실하게 찾을 수 있도록 주제와 관련된 문서 집합에서 찾아내는 과정을 의미한다.

man 페이지

유닉스에서 이어져온 초기 ’presentation’ 마크업이며 가장 일반적인 형식이다. man(1) 명령은 페이져와 아주 오래된 검색도구를 제공한다. 이미지나 하이퍼링크, 인덱싱은 지원하지 않는다. 인쇄를 위한 포스트스크립트 렌더링은 매우 잘된다. HTML로는 표현할 수 없다.(필요하다면 텍스트로만 가능하다.) 관련도구는 리눅스 시스템에 이미 설치되어 있다.

man 페이지 형식은 명령어의 요약이나 숙련된 사용자의 기억을 되살려주는 간단한 참조용 문서로는 나쁘지 않다. man 페이지는 복잡한 인터페이스와 많은 옵션을 가지려는 경향 하에서 삐걱거기 시작했다. 그러므로 이를 이용하여 많은 교차-참조를 갖는 문서를 관리하려고 한다면 (하이퍼링크의 지원이 없는 상황에서) 완전히 실패할 것이다.

HTML

1993-1994 이후로 웹이 나타나면서 HTML의 사용이 증가하였다. 마크업은 부분적으로 ’structural’ 이고 대부분은 ’presentation’이다. 어떤 브라우져에서도 브라우징이 가능하며, 이미지와 하이퍼링크도 잘 지원한다. 내장된 인덱싱 도구는 기능이 제한적이지만, 좋은 인덱싱 및 검색엔진 기술이 존재하고 현재 널리 쓰이고 있다. 인쇄를 위한 포스트스크립트 표현도 잘 된다. HTML 도구는 현재 전 세계적으로 사용되고 있다.

HTML은 여러가지 종류의 문서화에 매우 융통성 있고 적합한 방식이다. 실제로, 융통성이 무척이나 크다; 많은양의 마크업이 문서의 structure보다는 presentation을 표시 함으로 인하여 자동적으로 인덱싱하는데 문제가 생기면 man 페이지 형식을 공유하기도 한다.

Texinfo

Texinfo는 자유 소프트웨어 재단(FSF)에서 사용하는 문서화 방식이다. Texinpo는 강력한 Tex 포맷팅 엔진 위에 올라가는 매크로의 집합이다. 대부분이 ’structural’ 정보이고, 일부가 ’presentation’과 관계되어 있다. Emacs나 독자적인 **info**라는 프로그램으로 브라우징 할 수 있다. 하이퍼링크를 잘 지원하지만 이미지에 대한 지원은 하지 않는다. 인쇄와 온라인 형식을 위한 인덱스를 둘다 지원한다. Texinfo 문서를 설치하면 시스템의 모든 Texinfo 문서를 리스팅하는 ’dir’문서에 위치가 자동적으로 추가된다. 포스트 스크립트를 완벽하게 지원하며, HTML로도 사용할 수 있다. Texinfo 도구는 대부분 의 리눅스 시스템에 내장되어 있으며, Free Software Foundation (http://www.gnu.org) 에서 찾을 수 있다.

Texinfo는 설계가 잘 되어있고 인쇄용 책이나 크기가 작은 온라인 문서에는 유용하게 사용할 수 있지만, HTML과 마찬가지로 양서류과에 속한다. 즉, 마크업이 부분적인 ’structural’과 ’presentation’으로 구성 되어 있으며 ’presentation’ 부분은 렌더링을 할 때 문제가 있다.

DocBook

DocBook은 SGML을 기반으로 하는 크고, 정성들여 만든 마크업 형식이다. (XML보다 최근 버전이다.) 앞에서 소개한 표현 방식들과는 달리, 모두 'structural'을 표현하는 마크업으로만 구성되어 있고 'presentation'은 없다. 이미지, 하이퍼링크를 완벽하게 지원하며, HTML로의 렌더링이나 인쇄를 위한 포스트스크립트도 잘 지원한다.(인쇄의 질은 도구의 발전에 따라 증진될 것이다.) 이와 관련된 도구와 문서는 DocBook website (http://www.docbook.org/)에서 찾을 수 있다.

DocBook은 크고, 복잡한 문서 작성을 위한 훌륭한 방식이다. DocBook은 기술 매뉴얼의 작성과 이들을 다양한 출력형태로 렌더링하기위해 특별히 설계되었다. 이것의 단점은 복잡도와 도구들이 완전히 성숙하지 않았다는 것(비록, 빠르게 발전하고는 있지만)과 소개하는 수준의 문서가 부족하고 (아주 빈번하게) 매우 혼란스럽다는 점이다.

7.2. 미래의 문서화 방법

2000년 7월, 주요한 오픈-소스 프로젝트 그룹(GNOME, KDE, 자유소프트웨어 재단, 리눅스 문서화 프로젝트, 오픈-소스 발기그룹을 포함하는)의 대표가 모여서 수뇌 회의를 캘리포니아의 몬터레이에서 개최하였다. 회의는 문서화와 문서교환의 표준형식을 연구하고 확정해서 보다 풍부하고 통합된 형태의 문서화 형식이 발전할 수 있도록 하는데 그 목적이 있었다.

구체적으로, 회의 참석자 모두가 동일하게 인정한 사항은, 설치하는 즉시 시스템의 모든 문서 인덱스에 통합되어, 모든 문서가 단일한 인터페이스와 유닛 단위의 검색을 통해 브라우징이 가능하도록하는 문서화 패키지를 만드는 것이다. GNOME과 KDE는 단계적으로 이미 그러한 방향을 채택하고 있었으며, 이를 위해서는 'presentation' 보다는 'structural'의 마크업 표준이 필요하다는 것을 이미 이해하고 있었다.

회의에서는 명백한 문서화 경향을 승인하였다. 주요한 오픈-소스 프로젝트는 이미 문서화 형식으로 주로 Docbook을 적용하고 있거나 이미 적용하였다.

참가자들은 문서의 인덱싱을 지원하는 'Dublin Core' 메타데이타 형식(디지털 자료의 인덱싱과 관련하여 도서관 관리자들이 개발한 국제 표준)를 사용하기로 결정하였다. 여기에 대한 세부사항은 계속 해결해 나가고 있으며, 아마도 결과물은 DocBook 문서에, Dublin Core 메타데이터의 내장을 지원하기 위한 마크업이 추가되는 것으로 나타날 것이다.

방향은 명백하다; 인덱스 태깅과 Dublin Core 메타데이터를 기반으로하여 Docbook 문서의 자동 인덱싱을 지원하는 보조 표준과 함께 Docbook을 사용하는 것이다. 여기에는 여전히 빠진 사항이 몇 가지 있지만 그것들은 언젠가는 채워질 것이다. 예전의 'presentation' 기반 마크업을 쓸 날들은 얼마 남지 않았다.(이 문서도 2000년 8월에 Docbook으로 이전하였다.)

따라서, 새로운 오픈-소스 프로젝트를 시작하는 이들이 DocBook을 주 형식으로 해서 시작한다면, 변화를 앞질러 가면서 나중에 닥칠 고약한 변환 작업을 피해 갈 수 있을 것이다.

8. 홍보 방법

그 존재를 사람들이 알 수 없다면, 당신이 만든 소프트웨어와 문서는 세상에 도움이 될 수 없을 것이다. 또한, 인터넷에 프로젝트의 존재를 보여주는것은 사용자와 공동 개발자를 모으는데 도움이 될 것이다. 여기 그렇게 하는 일반적인 방법이 있다.

8.1. c.o.l.a와 Freshmeat에 발표해라

새로운 공개(release)를 comp.os.linux.announce" (news:comp.os.linux.announce)에 알려라. 이곳은 많은 사람들이 읽을 뿐만 아니라 "Freshmeat" (http://www.freshmeat.net)같은 웹기반의 what's-new 사이트의 중요한 소스이다.

8.2. 주제와 관련된 newsgroup에 발표해라

프로젝트와 직접적으로 관련된 주제의 USENET 그룹을 찾아서 그곳에 발표해라. 코드의 기능**(function)**과 직접 관련된 곳에만 게시하고, 관련없는 아무 곳에나 게시하는 것은 삼가야 한다.

예를 들어 IMAP servers 에 관련된 프로그램을 Perl로 작성해서 발표할 경우, comp.mail.imap에 확실히 게시해야 하겠지만, 새로운 Perl 기술의 교육적인 예가 아니라면 comp.lang.perl에 게시해서는 안될 것이다.

발표할 때에는 반드시 프로젝트의 웹사이트 주소를 포함시켜야 한다.

8.3. 웹사이트를 운영해라

프로젝트에 참여하는 견고한 사용자와 개발자 집단을 만들고 싶다면 반드시 웹 사이트를 가져야 한다. 웹사이트는 일반적으로 다음과 같은 요소를 포함해야 한다:

- 프로젝트의 개요(왜 만들었는지, 누구를 위한 것인지 등등)
- 프로젝트의 소스를 다운받을 수 있는 링크
- 프로젝트의 메일링리스트에 가입하는 방법
- FAQ 리스트
- HTML로 작성된 프로젝트 문서
- 관련되거나 경쟁관계에 있는 프로젝트의 링크

어떤 프로젝트는 아무나 마스터(master) 소스에 접근할 수 있는 URL을 보여주기도 한다.

8.4. 프로젝트의 메일링리스트를 운영해라

프로젝트 참여자가 정보를 교환하고 패치(patch)를 교환하기 위한 개인적인 개발자 리스트를 가지는 것이 일반적이다. 또 프로젝트의 진행 상황을 궁금해하는 사람들을 위한 공개 리스트를 가지는 것도 좋다.

예를 들어, 'foo'라는 이름의 프로젝트를 실행한다면 당신의 개발자 리스트는 'foo-dev'나 'foo-friends'가 될 것이다. 그리고 공개 리스트는 'foo-announce'가 될 것이다.

8.5. 중요한 아카이브(archive)에 배포해라

지난 몇년 동안 Metalab archive (http://www.metalab.unc.edu/pub/Linux/)는 리눅스 소프트웨어의 가장 중요한 교환 장소가 되었다.

방문자가 급격히 증가해왔다. 이 사이트는 보이는 것처럼 단순한 아카이브나 배포본 사이트가 아니다. 여기서는 오픈-소스 프로젝트를 진행하길 원하는 그룹을 위한 완전한 도구 - 사이트와 아카이브 제공, 메일링 리스트, 오류 추적, 채팅 포럼, CVS 저장소 등 - 를 전부 무료로 지원하는 프로젝트 호스트 서비스이다.

그 밖의 다른 중요한 곳의 주소이다:

- Python Software Activity (http://www.python.org) (Python으로 제작하는 경우)
- CPAN (http://language.perl.com/CPAN), Comprehensive Perl Archive Network(Perl로 제작하는 경우)

9. 프로젝트를 관리하는 방법

모두 자발적인 개발자들로 구성된 프로젝트를 성공적으로 관리한다는 것은 일종의 특별한 도전이다. 프로젝트 관리 방법은 HOWTO에서 다루기에는 너무나 큰 주제이다. 하지만 다행스럽게도 중요한 문제들을 이해할 수 있도록 도와주는 유용한 백서들이 존재한다.

기초적인 개발조직에 관한 토론과 빠르게-자주 공개하는 '시장 모드'에 관해서는 The Cathedral and Bazaar (http://www.tuxedo.org/~esr/writings/cathedral-bazaar/)를 보라.

동기 심리, 커뮤니티의 관습, 불협화음 등에 관한 토론은 Homesteading the Noosphere (http://www.tuxedo.org/~esr/writings/homesteading/)를 보라.

경제와 적절한 비즈니스 모델에 관한 토론은 The Magic Cauldron (http://www.tuxedo.org/~esr/writings/magic-cauldron/)을 보라.

이 백서들이 오픈-소스 개발에 대한 최종 견해는 아니다. 그러나 문서화한 최초의 진지한 분석이며, 아직은 이들을 대신할만한 어떠한 것도 나와 있지 않다.(저자들은 언젠가는 다른 생각이 나오기를 기대함에도 불구하고.)