Platforms Languages and Packaging

Dave Tarrant, Les Carr,

davetaz,lac@ecs.soton.ac.uk

Electronics and Computer Science

Platforms

- A Platform is a combination of Hardware and Software.
  - Windows/x86
  - Linux/amd64
  - iPhone/ARM
- Platform is also an environment in which software can interact with other things:
  - E.g. a webcam, GPS, or files

Platforms

- All have different APIs
- All have different capabilities and Limitations
- Can come in many combinations
- Can be freely configured to a limited or unlimited extent

Languages

- Each platform will prefer a different set of software tools which can take full advantage of all functionality
- .NET for windows
- C, C++ for Linux
- Objective C + COCOA for OS X

Cross Platform

- In order for something (including websites) to be considered cross platform it must function on more than one computer architecture or operating system.
- Time Consuming Task when the Platforms are so varied.
- Software written for an operating system might not work on all architectures.

Choose the right language

- Things like C and C++ are generally the best languages for cross-platform. Will still need to cross compile.
- Keeps the application low level, fast and with direct access to the platform.

Choose an Interpreted Language

- Programming language in which programs are 'indirectly' executed
- Can still compile code to be “native”.
- In many cases there is little performance difference between an interpretive- or compiled-based approach.

Interpreted Languages

- Can be compiled into machine code
- Can just be run, e.g. php, python
- Can require a framework/runtime environment to just-in-time compile
- Java is a good example of an interpreted language requiring JIT compilation and the presence of a JVM. Time to start JVM is often a killer for apps

Interpreted Langauges

- Mostly have to be compiled for full cross platform and even then only a limited number of platforms may be supported.
- Abstract the platform in many cases so loss of full low level API access is lost.
- Do your research before using!

Strategies

- Choose a Platform and Stick with it
  - Simple, Limited Market, All your eggs in someone elses basket
  - Very hard to switch!
  - No knowledge expansion

Multiple CodeBases

- Most complicated
- Most Expensive
- Most Complete
- Build a platform for Plug-ins
- Example is Microsoft Office which is actually 2 different versions.

Other Platforms

- The Web
  - HTML5, Java-Script, JSON, REST, HTTP CRUD
- Flash
  - If you don’t use flash for video (Not Mobile)
- If you had to choose one which would it be.

Choosing Your Platform

- NOT based upon your experience
- Based on intended user experience
- Don’t:
  - Make a location based application on a desktop
  - Make an application “online only” unless clear
  - Make a cross platform application if it can’t reach your primary market.

Platform Diversity

- Product testing of complex products becomes difficult when the market is so diverse
- Provides a demand for “support” services.
- Open Android Market vs Closed Apple Market.
  - Both have a strong API but diversity of devices and operating system support causes confusing
  - Conversely, when a problem hits an IOS device the whole community is affected, support is greater.
  - On Android the user could feel isolated.

Discussion: User Experience

- Pick a number of competing products (e.g. tablets) and discuss the different user experiences provided by each.
  - IS IT Google vs Apple?
  - OR IS IT Samsung, Amazon, Motorola etc (using Android) vs Apple
- Are these experiences consistent across all devices.
  - Which is the best Android tab experience, does it matter to the user, how will it affect your product?
- Does the “intended” functionality of the device (think Kindle Fire) change the expected user experience.

Summary

- Hard choices have to be made early
- These should not just be based on your own knowledge (e.g. I know java)
- Should be based upon the needs of the users and the platform with which they are familiar.
- Choosing the right target platform is key

Commonalities

- The Web (as both a platform and part of the environment)
  - More on web technologies will be introduced later in the course
- Source Code Control
- Packaging
  - Each platform will have a standard/prefered mechanism for ‘one click’ install.
- “App Stores”
  - A place for packages
  - Package repositories with pretty front ends which make money.
  - Package Repositories have existed in Linux for years and are still more advanced than the commercial ones.
  - Your coursework looks at one such example and tasks you with building a package for inclusion in a package repository.

Software Development Essentials

- Source Code Control
- README

- Ticketing System
- Changelog
- Packages & Downloads (not a zip)

Source Code Control

- Fill this slide full of pictures of protocols, servers, services.
- svn
- git
- bzr
- perforce
- hg
- cvs

Source Code Control

- What is a tag?
  - A “tag” references a specific point in history, e.g. an important point. Most use tags for version releases.

- What is a branch?
  - A parallel development of the software, not a point in history, e.g. experimental vs. stable. Code is then “merged”.

GIT

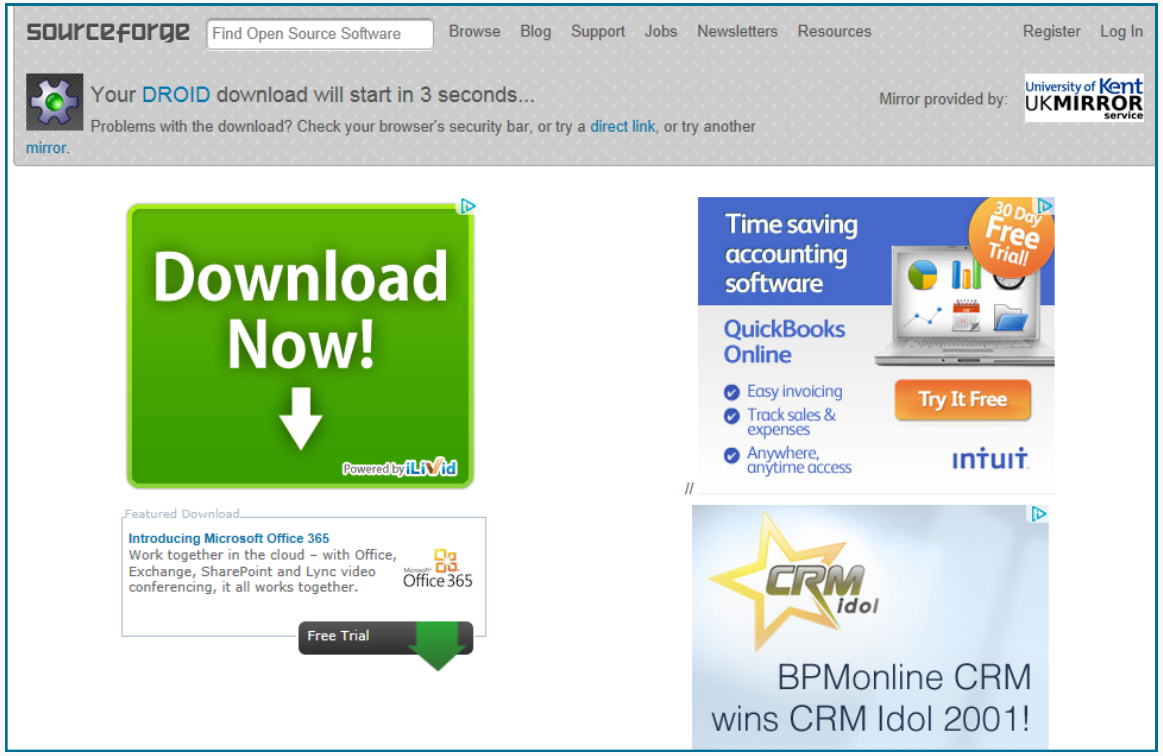
- Distributed
- Not dependent on network access
  - Great for coding on the train!
- Strong Support for non-linear development
  - branching, merging
- Scalable
  - Does not slow as the project history grows
- Staging
  - Allows customized commit operations, e.g. a specific set of files, sections of files. An entire commit can be previewed.

Source Code and Tickets

- Source code systems also contain many good issue tracking systems.
  - Fixes #xxx

Source Code Systems and Marketing

- Source code systems are typically not the best places for marketing a product

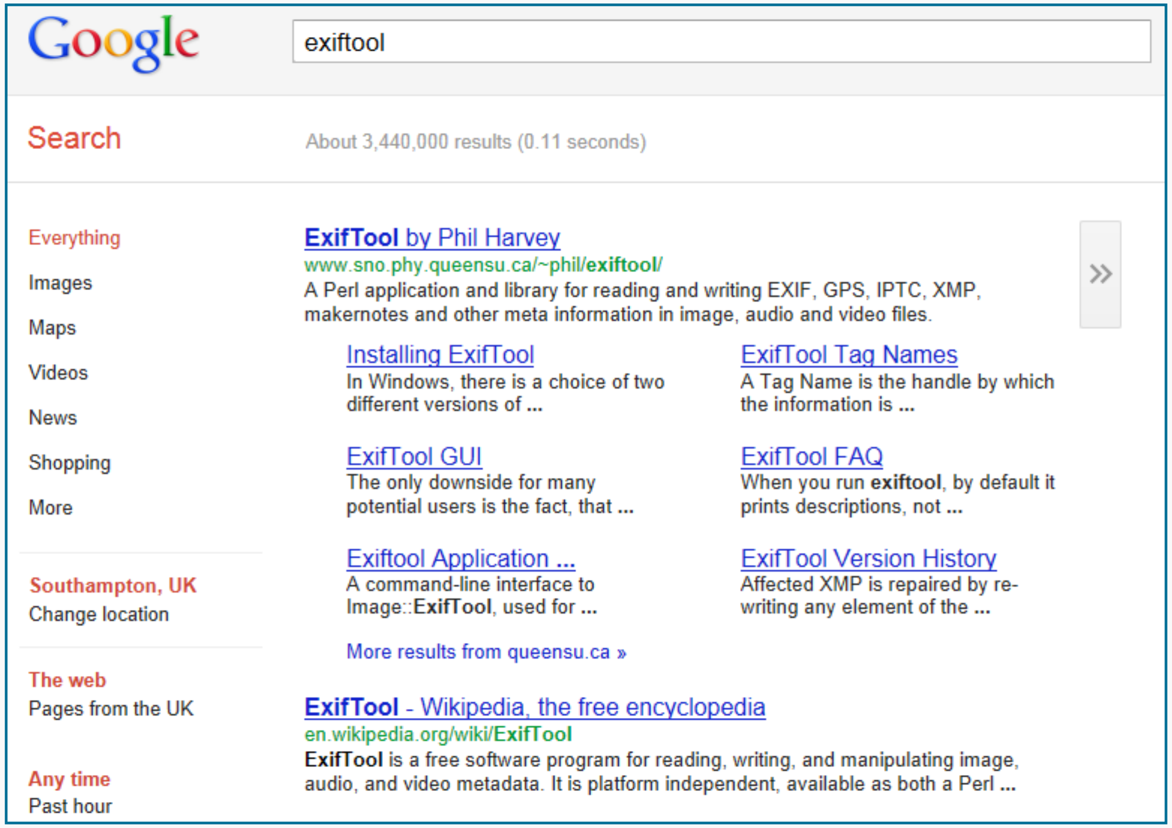


Your DROID download will start in 3 seconds...
Problems with the download? Check your browser's security bar, or try a direct link, or try another mirror.

Featured Download
Introducing Microsoft Office 365
Work together in the cloud – with Office, Exchange, SharePoint and Lync video conferencing, it all works together.
Microsoft Office 365

BPMonline CRM wins CRM Idol 2001!

Finding Tools – Exiftool



About 3,440,000 results (0.11 seconds)

ExifTool by Phil Harvey
www.sno.phy.queensu.ca/~phil/**exiftool**/
A Perl application and library for reading and writing EXIF, GPS, IPTC, XMP, makernotes and other meta information in image, audio and video files.

Installing ExifTool
In Windows, there is a choice of two different versions of ...

ExifTool Tag Names
A Tag Name is the handle by which the information is ...

ExifTool GUI
The only downside for many potential users is the fact, that ...

ExifTool FAQ
When you run **exiftool**, by default it prints descriptions, not ...

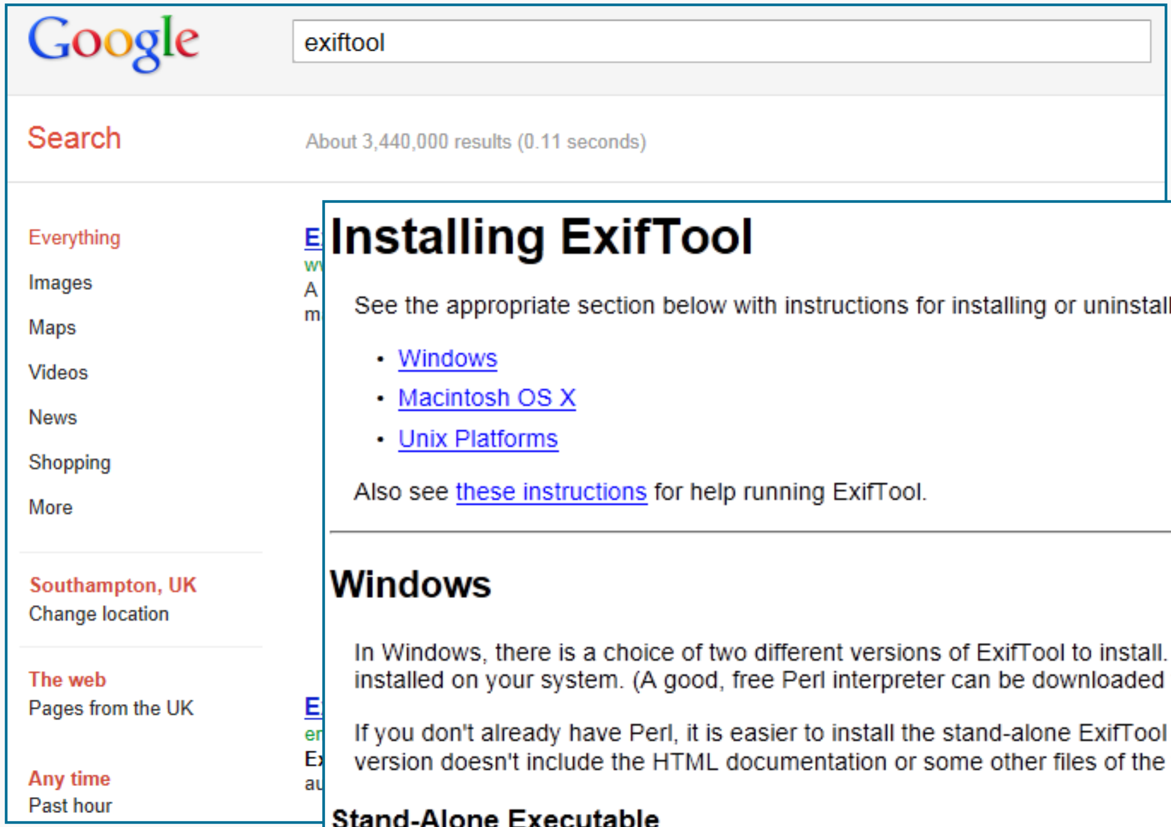
Exiftool Application ...
A command-line interface to Image::**ExifTool**, used for ...

ExifTool Version History
Affected XMP is repaired by re-writing any element of the ...

More results from queensu.ca »

ExifTool - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**ExifTool**
ExifTool is a free software program for reading, writing, and manipulating image, audio, and video metadata. It is platform independent, available as both a Perl ...

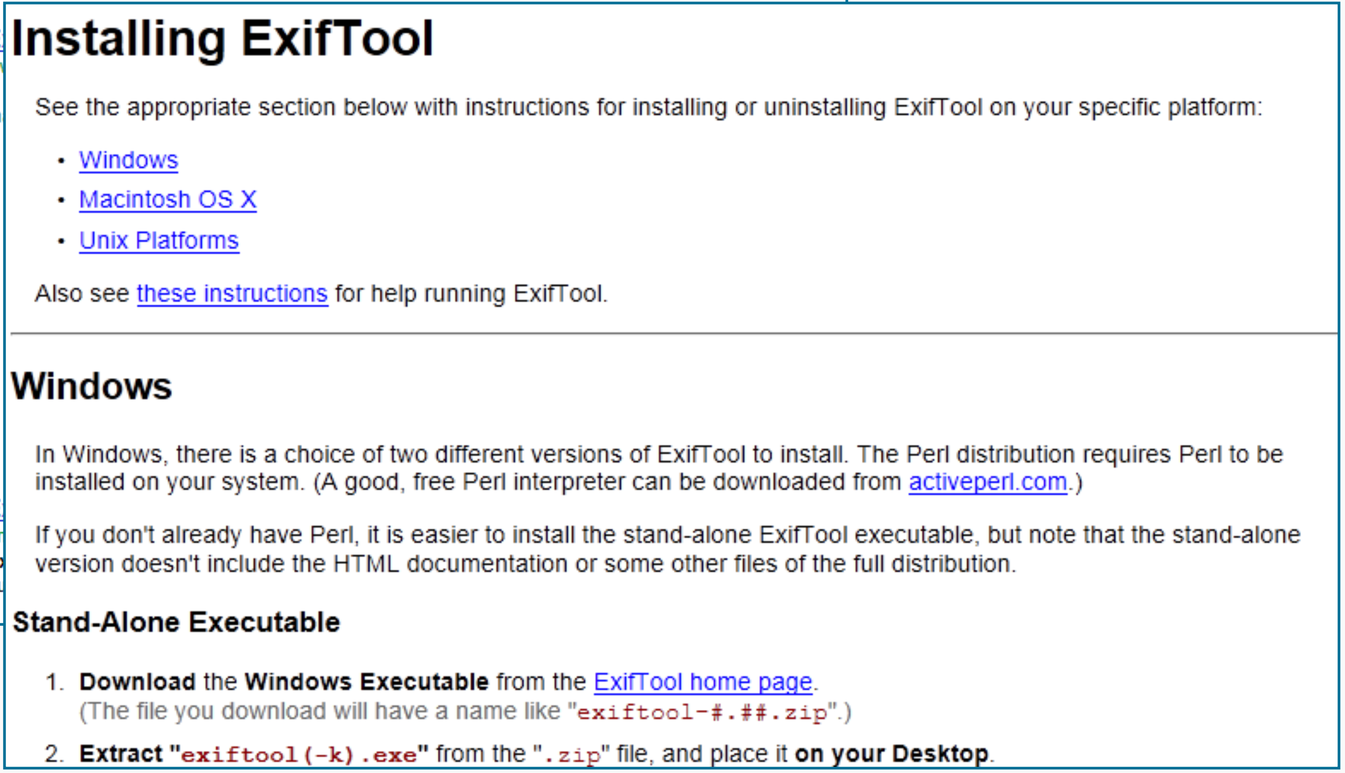
Finding Tools – Exiftool



exiftool

Search

About 3,440,000 results (0.11 seconds)



Installing ExifTool

See the appropriate section below with instructions for installing or uninstalling ExifTool on your specific platform:

- Windows
- Macintosh OS X
- Unix Platforms

Also see these instructions for help running ExifTool.

Windows

In Windows, there is a choice of two different versions of ExifTool to install. The Perl distribution requires Perl to be installed on your system. (A good, free Perl interpreter can be downloaded from activeperl.com.)

If you don't already have Perl, it is easier to install the stand-alone ExifTool executable, but note that the stand-alone version doesn't include the HTML documentation or some other files of the full distribution.

Stand-Alone Executable

1. **Download** the **Windows Executable** from the ExifTool home page.
   (The file you download will have a name like "`exiftool-#.##.zip`".)
2. **Extract "`exiftool(-k).exe`"** from the "`.zip`" file, and place it **on your Desktop**.

Marketing Your App

- Make sure you can FIND it
- Give away preview copies
- Introductory Pricing
- Market it! Tell as many people as possible without annoying them.
- Make it clear and pretty!

Summary

- Use source code control properly
  - Write useful commit messages
  - Make commits small so they can be reversed
  - Make one commit to close a ticket, this will automatically link to the ticket in most systems
  - Tag working versions for packaging
- Package you App for easy installation
- Realise that developers and users have different needs, want to see different web sites.

Package Platforms

- Linux Platforms – Most Mature
  - Dependencies!
  - Upgrade and config management
  - Manuals
  - Examples
  - Easy to make mash ups!
- Windows/Apple Apps
  - Apps are Sandboxed
  - Can require other packages but management is not clear
  - Fine for most things, not brilliant as servers.

CW1

- CW1 – Packaging your software (Individual)
  - Use the reference code to build a debian/ ubuntu package.
  - Must use Revision Control (this will give you a changelog)
  - Should use all other pacakging features when applicable:
    - Man pages
    - Examples
    - Documentation

Building Pacakges

- Debian/Ubuntu packaging uses many config files to build a binary package
- Redhat/Ferdora uses a single spec file
- IOS/Android/WinPhone integration less clear have to use specific tools.

Building Pacakges

- All the systems are demanding and very fussy about requirements.
- This is done for a reason, they all called package managers, so need to remain manageable.
- The Apple App Store is not the only platform where reviews take place.
  - Debian requires a package sponsor.

File Structure

- debian
  - control
  - copyright
  - changelog
  - rules
  - dirs
  - preinst/postinst
  - prerm/postrm

Part1: control

- The Package Management control file:
  - Name
  - Type
  - Priority
  - Maintainer
  - Dependencies!

Part2: copyright

- Lists the copyright and license of the upstream sources (i.e. the software)
- If using a supported license, these are already installed on the platform thus don’t have to be listed in full.

Part 3: changelog

- A changelog is one of the most useful ways to communicate. It stays with the software.
- Strictly speaking the deb.rpm changelog is about changes to the package, not the software provided by the package, often confused however.
- A changelog is better than no changelog.
- In GIT you can't differentiate the 2 easily, hense why both paradigms are common.

Part 4: rules (deb)

- Defines how to build and install the package from source.
- Basically another Makefile which is very debian specific.
- Handles:
  - Changelog
  - Man Pages
  - Config Files
  - Examples
  - udev
  - …

Part5: dirs (optional)

- Specifies needed directories not created as part of install. e.g. they exist already.
- These directories may not exist in fakeroot so listing them here is an easy way to cheat fakeroot.

```
usr/share/software_name
etc/apache2/sites-available
```

preinst/postinst

- Scripts to run prior to installation and after successful installation.

- Can check system state for compatability.
- Can perform enable actions, like setting up a database or restarting a service.
- Are also run on upgrades.

- Files have many conditional sections, like a Makefile.

http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html

prerm/postrm

- Same as preinst/postinst but run when removing a package

- In most cases these should perform the opposite of the preinst/postinst scripts leaving the system in a clean state from which the installation scripts can be run again.

http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html

Getting Started

- To create a set of template files:
  - Create a directory call my_app-1.0.0
  - Change into this directory
  - dh_make --help
  - dh_make –s –n … (other options)
- These can then be taken and used in your package
- dh_make can also be used on a specific source code tarball.

Deb-Helper

- Very useful commands which just do stuff for you
  - dh_installman
  - dh_installdocs
  - dh_installexamples
  - dh_testroot
- E.g. dh_installman installs any files listed in *package*.manpages into the correct manpage location on that platform.

Follow the rules.

- The rules file is key
- Put testing in here (using more of the dh_commands)
- These tests will throw very fussy, but accurate errors

Use Lintian

- Lintian dissects Debian packages and tries to find bugs and policy violations.
- It contains automated checks for many aspects of Debian policy.
- Checks for common errors
- Package will not be sponsored (or get as many coursework marks) if it is not “Lintian Clean”

Summary

- Package files only need to be written once (but kept up to date for dependencies etc)
- Be careful to separate software build from package build requirements, the two are different.
- Use the deb-helper scripts
- Use Lintian.
- Fulfil all the requirements and more as you will feel much better later.
- Enjoy the time when it first compiles as a package!