New Developer's Guide

Barış Metin <baris@pardus.org.tr>, A. Murat Eren <meren@pardus.org.tr>

29 Mayıs 2006

Translation: K. Deniz Öğüt

Proofreading: Shane Shields, Görkem Çetin

Özet

This document describes what a new developer should know and do to join the development process of Pardus project.

İçindekiler

1 Introduction

In this document we'll try to explain how contributers can support the Pardus Liux, which is developed under the structure of Pardus Project, and the requirements to be a Pardus developer. Although we are talking about the Pardus development model in particular, we believe that the requirements explained in the document apply to almost all distribution projects.

Please feel free to browse the web pages of Pardus for other information not contained in this document. From those pages you may find the information needed on other documents, tools used and services. Pardus pages can be reached from `http://www.pardus.org.tr`

2 What is a developer?

To put it roughly, it is not wrong to say that the project should fulfill two types of mission. As it is the case for any distribution, gathering the software and preparing the infrastructure to enable this task and sustaining it, as well as developing new tools and technologies to form the soul of the distribution are among the required tasks.

When referring to *developer* we describe not only those who write programs but also those who fulfill any single task. We are aware that work has to be done in many areas such as documentation, bug controls, visual materials, translations etc. besides programming to implement a software project. We need all of them for Pardus as well.

How will I start?

First of all its a good idea to keep up with current events. To do this its mostly effective to observe for a while and watch how work proceeds. It might be useful to join the e-mail lists which can be accessed from the Pardus web pages and follow the discussions, ongoing events, investigate bug reports, the solutions proposed regarding the bugs and examine the documents published.

By considering the outlined methods you may continue to help development. You can add your comments and proposals for solutions to the bug reports by testing the sofware or you can report the new bugs you found. You can contribute to develop innovative technologies and add new features or you can support a step of the process which you think is not working (maybe working too slow) and speed it up. By doing so, you should always be in communication with other developers and contributers. As its true for all tasks performed by more than one person, we need to have knowledge of each other.

Our needs can be summarized as the articles below:

2.1 Developing software and debugging

You can contribute your knowledge and labour on developing software by assisting with cleaning up the source and helping innovative technology teams working in this field. They can provide the best information on how you can help them. The Pardus e-mail lists which you can join via `http://liste.pardus.org.tr/mailman/listinfo` will be helpful.

You may examine the reported bugs, and find solutions for them, using Pardus bug tracking system which can be accessed via the web pages.

2.2 Translation

Localization of free software, to enable them to be used with correct and perfect Turkish is another requirement for us. In this context, you may work with localization teams organized for almost all major free software project. We'll draw direct benefit from your support for localization projects if you give priority to the software selected by Pardus project.

To say in short, support localization efforts. Beyond rough translations try to improve the quality of translations. While supporting translation works, by providing not only Ulusal Dağıtım but also all existing Linux distributions with packages in problem-free Turkish, you will be part of an important mission.

To support notable translation groups, you can reach the localization works you feel interest in via `http://cekirdek.pardus.org.tr/~baris/tmp/ceviri-calismalari.html` (in Turkish)

2.3 Tests and bug reports

You can report bugs by testing the software we choose. You can controle the bugs in other distributions and report the situation. To know that the very same bug exists in other distributions might help us to produce a proper solution. If that problem is solved -or never existed- in a particular distribution, reaching a solution will be faster by examining the work that particular distribution did for that software. You can reach our bug tracking system via `http://bugs.pardus.org.tr/`

2.4 Graphical Design, Multimediea

If you are talented in this domain, you can help with the topics which require graphics knowledge, such as the icon sets, font types and colour themes. Feel free to ask for help from us via e-mail lists when you need visual materials to use in your work.

2.5 Documentation

You can support the documentation of the ongoing projects. Along with the user's documents you can prepare "How To" documents for the developers that have newly joined the project. You can assist us to update our Web pages or you can help us to keep foreign language -such as English, German, Spanish- translations of our pages up-to-date.

2.6 Publicity

You can support the publicity of the project and encourage more people to learn about it.

Probably the list of the topics you can help is not limited with those above. By rendering your opinions on the discussion points within e-mail lists, advancing proposals, providing the lists with constructive criticism and developments related with the distribution, you can find new topics to help.

3 What is the responsibility of the developer?

Indeed, the answer of this question might vary considerably depending on the subject you begin to work on, but we can mention the basic responsibilities of all developers in general. In fact, the responsibilities described here are basic points for many developer to work together in harmony. In this sense, although what is mentioned below is true in general, you can find your own way to cooperate as you become familiar with your working environment and friends.

3.1 Continuity

A developer should be prepared to work on the projects s/he is dealing with continiously. Here "continuity" doesn't stand for a 7/24 work but it stresses the continuity of the subject to be dealt. For different areas are clannish in an operation system, developers should keep up with each others speed.

3.2 Accuracy

In the project you might be working on the same subject with more than one developer. What's more, there might be other developers affected by your work. For this reason not only the continuity of your work but also the accuracy are of great importance in proportion with the extend they affect other developers. If you are performing your work on one of the main development repositories, you should take into account the other developers working with you that are affected by your work, in all stages. You should make sure that any change you make does not -even slightly- prevent other developers from doing their job.

3.3 Determination

If you have the power of decision for a subject you are working on, changing your decisions too often makes it difficult for the other developers to follow and what's more important it makes the developers who are dependent on you to have difficulties while working. Hence it will be useful to take well-thought decisions and to ask for the opinions of the other developers before making a decision.

At a particular point if you feel that you should change your decision anyway, you can do this by letting the other developers know about it and -if your previous works are already being used- by working in an "experimental" area harmless to them.

3.4 Communication

It'll be useful to let other developers know about the decisions you make, the steps you take and the changes -even the minor ones- you make. In this way, you can adapt new developers -who are always needed- to the subject you are dealing with easier, you can cooperate with the developers you work with more consistently and faster. When you'll be in need for help in the future, with the help of this attitude, you can be sure that a solution will be produced faster because another develepor already knows what you want to do. For communication you can use e-mail lists, the documents you'll prepare or explanatory information (such as subversion messages) you'll add to the changes you make.

4 New developer application

There's always room for a new developer. You too can apply to be a Pardus developer and become an official developer as well. Although it's a prerequisite to accept the responsibility of being a new developer, it's not enough for all cases. On the other hand, we can put it openly that we are enthusiastic for delegation of responsibility and accepting new developers. For details please keep on reading...

4.1 Who may submit?

Everyone who is working for Pardus and who accepts the developer responsibilities described in this document may submit a *new developer application*. The main channel of communication is the e-mail lists. In order to accept an application, applicant's determination and style of working should be obverved. In this sense, you can select a task suitable for you and begin working on it before you submit. By sharing your work with the other developers you enable them to examine your work and so other developers will have the chance to become acquainted with you.

For example, it may be a good starting point to work on the software which is already in the package repositories, propose new patches and solutions, examine and test other solution proposals and report what you get.

Besides, there's a quality e-mail list [1] in which those who would like to support the project can introduce themselves and pick a task suitable for them.

4.2 How to submit?

To submit the application you can e-mail to basvuru@pardus.org.tr

In your application e-mail;

1. the development subjects you are working on related with Pardus,
2. other subjects you want to work on,
3. the e-mail address/username which you use for the bug tracking system,
4. your areas of specialization,
5. a Pardus developer who knows about your work and might be a reference for you

should be included.

You should include the username and ciphertext (created by htpasswd) of your password in your application e-mail. By doing so, you provide us with the ciphertext of your password where you are the only one who knows the plaintext. We can then add your username and password wherever an identitiy check is required.

4.2.1 to create a password with htpasswd

You can create your password and username by using htpasswd program and you can attach the output file to your e-mail. To do this, you can use the command below:

```
$ htpasswd -c password_file user_name
New password:
Re-type new password:
Adding password for user user_name
```

As a result an output file (password_file) will be created and you can attach it to your e-mail.

[1] Quality e-mail list can be reached from http://liste.pardus.org.tr/mailman/listinfo/kalite

4.2.2 to create a password with perl

To do this the, you can use the command below:

```
perl -e "print crypt('yourpassword','xy'),\"\n\";"[2]
```

This command will create just your password. In your e-mail you should mention the username you want to use.

4.2.3 to create a password with python

You can do the same with python by using the command below:

```
python -c "import crypt; print crypt.crypt('password', 'xy')"
```

In this case as well, just the shadowed form of your password will be given as output. In your e-mail you should mention the username you want to use.

5 Subversion repositories

The development process in Pardus is maintained via a Subversion version control system. Subversion is an open source version tracking system. Its a development infrastructure which makes it possible for more than one application developer to work together unconcerned about demolishing each others' changes. Thanks to this, any single software's development process can be tracked backwards, the changes made gradually can be watched and can be easily returned to any version of a particular time.

At the moment there are two subversion repositories we use in Pardus structure .

5.1 Pardus Repository

Pardus repository is the one in which the products being developed within the project are kept. All the software developed for Pardus is kept in Pardus repository. The repository is at the address https://svn.pardus.org.tr/pardus. It keeps the source package repositories where the software's PiSi packages are found.

More information on the Pardus repository can be found in the document Pardus Depo Politikası (Pardus Repository Policy).

[2]'xy' are two random characters given to crypt() function as a parameter to be used in shadowing.

5.2 Hierarchy of repository directories

Each Pardus subversion repository has the hierarchy of directories detailed below.

There are three main directories in a repository: trunk, tags and repos.

5.2.1 trunk/

Trunk is the directory in which continuous work is performed.

Each project module (document, Web pages, software projects, etc.) has its own directory under trunk/.

5.2.2 tags/

Tags directory is the place where, for any module, the work done under trunk is tagged and copied to. Within this directory, there exist 3 directories.

- **tags/RELEASE/**: This is the directory which software (or modules) use to tag their own version numbers. For example, version 0.2 of tasma is tagged in tags/RELEASE/tasma-0.2 directory.
- **tags/BLACKHOLE/**: Its a blackhole the projects for which the development is stopped (because of the lack of a developer to deal with it or no more need for that project) are "thrown" into. When the projects in it are desired to be used again, they are copied under trunk/ and worked on.
- **tags/RESTRUCTURED/**: Its the directory in which the module is put if the software (or module) undergoes a total restructuring and if the old files will not be used anymore. For example, "abc" project which was restarted to be written on 28th May 2005 is moved under tags/RESTRUCTURED/abc-2005-05-28/.

5.2.3 repos/

Repos is a directory in which developers can advance their work in an experimental nature without disturbing the other developers working on the same module under trunk. The developer can proceed her/his experimental work by creating her/his own directory under repos/.

(As a rule which is valid only for packeges repository, the documents related with this project are found under repos/doc directory.)

6 Using Subversion

A very detailed user's manual[3] is available. Besides that book, information about the project can be reached from the frequently asked questions page[4] on the own Web site[5] of Subversion project. In this section frequently needed commands for practical usage are tried to be told by examples.

6.1 How do I know if I have Subversion in my system or not?

As the quickest way to see if you have Subversion in your system or not, you can refere to the output of "svn –version" command. It's a good sign if you see something like this:

```
evreniz@jaco:~$ svn --version
svn, version 1.0.3 (r9775)
  compiled May 19 2004, 21:28:49
Copyright (C) 2000-2004 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet (http://www.Collab.Net/).
The following repository access (RA) modules are available:
* ra_dav : Module for accessing a repository via WebDAV (DeltaV) protocol.
  - handles 'http' schema
  - handles 'https' schema
* ra_local : Module for accessing a repository on local disk.
  - handles 'file' schema
* ra_svn : Module for accessing a repository using the svn network protocol.
  - handles 'svn' schema
evreniz@jaco:~$
```

If it doesn't exist, from `http://subversion.tigris.org/project_packages.html` you can have the package prepared for your distribution or operating system and install it. To our knowledge, all of the general Linux distributions, including Pardus, has subversion (svn) packages prepackaged.

6.2 What is a repository?

A repository is a disk area on which the last version, all the versions prior to the last version and the changes between versions of the software package(s) every developer works on, information including their user, date and cause are stored which can be reached by several methods.

[3]http://svnbook.red-bean.com/
[4]http://subversion.tigris.org/project_faq.html
[5]http://subversion.tigris.org/

6.3 How can I see what directories exist in a repository?

A single repository may contain more than one directory in it. The hierarchy of a repository is just like the inside of a directory on a disk. So, you can browse without having to copy all the repository to your disk and just get a view of the part you want to work on or have a look at. The list of directories and files in a repository is displayed using "svn ls repository_address" format:

```
$ svn ls http://svn.pardus.org.tr/uludag
repos/
tags/
trunk/
$ svn ls http://svn.pardus.org.tr/uludag/trunk
COMAR/
comar_prototip_old/
web/
$ svn ls http://svn.pardus.org.tr/uludag/trunk/COMAR
COMAR-1.sxw
COMARd/
CSL/
OM/
SlicerAPI/
confparser/
$ svn ls http://svn.uludag.org.tr/uludag/trunk/COMAR/confparser
GenericParser.py
README
branchedParser.py
comar_configparser.png
config_files/
confparser.py
flatParser.py
sectionedParser.py
$
```

6.4 How do I get a copy of a directory in the repository?

In order to create a copy of the repository "svn co" command is used. Once the copy is created, no more processing is performed on this command copy.

```
$ svn co http://svn.pardus.org.tr/uludag
A uludag/trunk
...
...
```

You can treat the repository as if its an URI. By doing so, you can get any subdirectory in the repository.

```
$ svn co http://svn.pardus.org.tr/uludag/trunk/COMAR
A COMAR/COMAR-1.sxw
A COMAR/CSL
...
...
```

6.5 How do I know if my copy is up-to-date or not?

You have to update the copy of the repository you have periodically with “svn update” command in order to know about the last changes and to track the last version. If you call the command alone the files in the directory you are on and the entire directory will be updated. Besides, you can add the address of the directory or the single file you want to update to the end of the command.

```
~/work/uludag/[...]/uludag/trunk $ svn update
U tasma/modules/tasmanet/device.cpp
U tasma/modules/tasmanet/devicesettings.cpp
U tasma/modules/tasmanet/device.h
U tasma/modules/tasmanet/devicesettings.h
Updated to revision 158.
~/work/uludag/[...]/uludag/trunk $
```

6.6 What do the signs by the files mean?

While you are working with SVN and during processes such as updating and searching, as in the previous example, the signs by the files are to inform you about what kind of a change related with the next file is performed.

One of the letters U, D, A, C or G may be found by files:

- **A** Added
- **D** Deleted
- **U** Updated
- **G** Merged (the last update you got from the repository is merged with the file you are performing local changes)
- **C** Conflicted (the last update you got from the repository is conflicted with the changes you performed localy)

6.7 I changed some files; what shall I do now?

You can use “svn status” command whenever you want to see what changes you made in your copy. This command can run with an URI you add to the end of it as all the other commands. Below its seen that a file is added to, a file is deleted from and two files are changed regarding the last updated copy of the repository:

```
~/work/[...]/trunk/COMAR/comar $ svn status
A COMARd/csl/degisiklik
D COMARd/csl/loader.py
M COMARd/COMARValue.py
M comar-call/rpc.c
~/work/[...]/trunk/COMAR/comar $ svn status COMARd/csl/COMARValue.py
M COMARd/COMARValue.py
~/work/[...]/trunk/COMAR/comar $
```

Also, you can learn what you particularly changed in changed files with “svn diff” command:

```
~/work/[...]/trunk/COMAR/comar $ svn diff comar-call/rpc.c
Index: comar-call/rpc.c
===================================================================
--- comar-call/rpc.c (revision 158)
+++ comar-call/rpc.c (working copy)
@@ -146,6 +146,7 @@
  if (len == 0) break;
  if (len == -1) {
          puts("connection broken too soon");
+         //totally different change
          break;
  }
  printf("RECV[%s]\n\n", buf);
~/work/[...]/trunk/COMAR/comar $
```

6.8 I added a new file but there is a “?” by it...

While you are working on the copy of the repository when you would like to create a new file, you should inform your local copy about your intention to add that file to the repository with the help of “svn add” (it has sister commands such as “svn copy”, “svn del” as well). Let’s explain why there’s such a need as follows: Let’s assume that you would like to compile and test an application which is in your local copy. In this case, some files that you don’t prefer to send to the main repository will be created in your work copy such as Makefiles and *.m4 files which only you have any need for.

In such cases, it will be very advantageous and convenient if the files added locally are not added to the repository as well, because when you change the source code of the software, recompile it and decide to send it to the repository at a proper time, you know that the other files are not going to the repository. With “svn add”, you add to the repository *the files you want to add*. “svn del” will not mentioned again.

```
~/work/[...]/COMARd/csl/sample $ svn status
~/work/[...]/COMARd/csl/sample $ touch newscript.csl
~/work/[...]/COMARd/csl/sample $ svn status
? newscript.csl
~/work/[...]/COMARd/csl/sample $ svn add newscript.csl
A newscript.csl
~/work/[...]/COMARd/csl/sample $ svn status
A newscript.csl
~/work/[...]/COMARd/csl/sample $
```

6.9 I want to revert the files I changed to their original states.

You can revert the changes you made to their original state as in the last copy using “svn revert” command anytime you like:

```
~/work/[...]/trunk/COMAR/comar $ svn status
A COMARd/csl/thechange
D COMARd/csl/loader.py
M COMARd/COMARValue.py
M comar-call/rpc.c
~/work/[...]/trunk/COMAR/comar $ svn revert comar-call/rpc.c
Reverted 'comar-call/rpc.c'
~/work/[...]/trunk/COMAR/comar $ svn status
A COMARd/csl/thechange
D COMARd/csl/loader.py
M COMARd/COMARValue.py
~/work/[...]/trunk/COMAR/comar $
```

Its also possible to revert all of the files to their originals recursively...

```
~/work/[...]/trunk/COMAR/comar $ svn revert . -R
Reverted 'COMARd/csl/thechange'
Reverted 'COMARd/csl/loader.py'
Reverted 'COMARd/COMARValue.py'
~/work/[...]/trunk/COMAR/comar $ svn status
~/work/[...]/trunk/COMAR/comar $
```

6.10 I want to send the files I changed; what to do?

If you are sure of the last condition of the files you changed, you can use "svn commit" command in order to transmit your changes to the repository. By using this command -as its true for all the rest- you can send to the repository a single file, a single directory and what is under it or all the changes you made. When you run "svn commit", svn -using your preferred text editor- opens a file for you with the changes you made listed in it in order to make sure others see what you changed and to enable your changes to be logged for backward tracking in the repository. To alter the text editor opened as the preferred one you can make use of the environment variable that svn uses named SVN_EDITOR:

```
~/work/[...]/COMARd/csl/sample $ SVN_EDITOR="vi" svn commit
~/work/[...]/COMARd/csl/sample $ SVN_EDITOR="mcedit" svn commit
~/work/[...]/COMARd/csl/sample $ SVN_EDITOR="kwrite" svn commit
.
.
```

As soon as you write the changes to the text editor, save what you wrote and close the editor, svn will begin to send the changes in your local copy to the repository.

6.11 How about the other commands?

You can run Subversion to learn Subversion commands as well. "svn help command_name" feeds you back with detailed information about *command_name* while "svn help" serves you with a list of commands you can use.

```
$ svn help
usage: svn <subcommand> [options] [args]
Type "svn help <subcommand>" for help on a specific subcommand.
Most subcommands take file and/or directory arguments, recursing
on the directories. If no arguments are supplied to such a
command, it will recurse on the current directory (inclusive) by
default.
Available subcommands:
add
blame (praise, annotate, ann)
cat
checkout (co)
cleanup
commit (ci)
copy (cp)
delete (del, remove, rm)
```

```
diff (di)
export
help (?, h)
import
info
list (ls)
log
merge
mkdir
move (mv, rename, ren)
propdel (pdel, pd)
propedit (pedit, pe)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
resolved
revert
status (stat, st)
switch (sw)
update (up)
Subversion is a tool for version control.
For additional information, see http://subversion.tigris.org/
```

In order to have detailed information about a Subversion command;

```
$ svn help add
add: Put files and directories under version control, scheduling them for
addition to repository.  They will be added in next commit. usage: add PATH...
Valid options:
--targets arg          : pass contents of file ARG as additional args
-N [--non-recursive]   : operate on single directory only
-q [--quiet]           : print as little as possible
--config-dir arg       : read user configuration files from directory ARG
--force                : force operation to run
--auto-props           : enable automatic properties
--no-auto-props        : disable automatic properties
```

commands can be used or Frequently Asked Questions can be browsed from `http://subversion.tigris.org/project_faq.html` or Subversion book can be read from `http://svnbook.red-bean.com`

7 Rules for using Subversion

Subversion repository is a common place shared by all developers. In order to work together developers should be using the repository efficiently, properly and in an organized way.

Rules for using Subversion are the rules to be respected by the developers having the privilege to write to Pardus repositories.

7.1 Always work with an updated repository

The updates on the Subversion repository will be more frequent as the quantity of the developers increase. In order to know about the rest of the process and to avoid a conflict between what you do and the rest to be done, before you begin to work, always update your repository by means of *svn update* command.

7.2 Think before you commit

Think twice before you commit[6] the changes you made to the Subversion repository. The data you commit to the repository will reach to all developers and affect their work. In this sense, its of great importance to follow the articles below.

1. Do not commit a non-running code to the Subversion repository.
2. Always update your repository before you commit by means of *svn update* in order to get the last changes. Be sure that the changes you made do not conflict with the others.
3. Pay attention to what you commit. To make sure of this, always control the changes you are about to commit by means of *svn diff* command before commitment.
4. Always test the changes you made. Even better, test them twice.

7.3 Add descriptive messages to your commitments

Explanation messages used in the commitments should focus on the change that is made and they should be as descriptive as possible. As much as you can, try to add explanation messages related with only the files you made changes on. However, in the limits of the context, you can include all the information which can not be derived from the output of a svn diff command in your explanation message.

Refraining from adding a proper explanation message will make it difficult to understand the changes you made.

[6]commit= the action performed by means of svn commit command

7.4 Abide by the work plans

If a work/time plan exists for the distribution in general or if the main developer of the component you are working on sets such a plan, abide by this plan regarding your commitments.

For example, an application developer might want to stop adding new features to the application at a particular time and might want to work on fixing the known issues. Its expected that the change you make is coherent with this rule.

If you are not sure of the coherency of the change you made with the plan, you have to refer the related e-mail lists or the main developer.

7.5 If you made a change affecting more than one component, inform all developers about the change.

In order to ensure all developers know about the "major" update you made, always send an informative message to the related e-mail list.

7.6 Take the responsibility of the change you made

If the update you made is creating a problem, take the responsibility of it and make sure to solve it yourself or by getting help.

7.7 Respect the generally accepted principles

Obey the general rules accepted during developers' discussions and be sure that your changes are not violating those rules. In any case you are unsure you can always choose the "communication" path.

7.8 Enter the bug number when solving a bug in the bug tracking system.

If the update you make is solving a reported bug, in order to synchronize the bug tracking system with the updates in the repository, notify the bug you solved and close the bug in the bug tracking system afterwards.

7.9 Update the files which you are responsible

Update only the files which are in your responsibility. If you find a bug in files which is in another developer's responsibility, first, discuss the situation either by directly contacting the responsible developer or by asking the other developers in e-mail lists and only after doing so make an attemp to update the repository. If the responsible developer does not accept the changes you made, behave respectfully.

7.10 Do not add the automatically created files to the repository

Do not add the files such as Makefile, Makefile.in, configure scripts, etc. which are created afterwards by compilation tools. These files will be recreated in different forms in all developers' machines and will be perceived as an update by the other develepors. In general, adding these files to the repository is perceived as a bug.

7.11 Perform atomic updates

Commit all the changes related with a particular improvement/update at once. Subversion lets you commit more than one file at once. Other developers might be confused because of seperate commits and they can miss the improvements you made.

Kaynaklar

[1] Metin, Barış & Onur, Çağlar (November 2004). Ulusal Dağıtıma Nasıl Yardım Ederim? http://www.pardus.org.tr

[2] Metin, Barış (November 2004). Paketler Deposu Yeni Geliştirici Başvurusu. http://www.pardus.org.tr

[3] Eren, A. Murat (November 2004). Subversion Deposu Kullanma Kılavuzu. http://www.pardus.org.tr

[4] Barth, Andreas (2005). Debian Developer's Reference. http://www.debian.org/doc/manuals/developers-reference/

[5] Fox, Tammy & Pennington, Havoc (2003). Fedora Project Developer's Guide. http://fedora.redhat.com/participate/developers-guide/

[6] KDE (2004). Applying For a KDE CVS Account. http://developer.kde.org/documentation/misc/applycvsaccount.php

[7] KDE (2004). KDE CVS Commit Policy. http://developer.kde.org/policies/commitpolicy.html