title:	How to Locally Branch a Project with Arch
license:	General Public License, V2
copyright:	2004, 2005 Canonical Ltd. ../../bazaar-docs/src/LocallyBranchingAProject.rst:3: (WARNING/2) Cannot extract compound bibliographic field "copyright".
Author:	James Blackwell <jblack@gnuarch.org>
original-location:	jblack@gnuarch.org--2004/bazaar-docs--initial--1.1

If you are reading this howto, then you have decided to create a local branch off of a publically available archive. In order to use this mini-howto, you are expected to understand how to resolve minor conflicts. In this case, you have become interested in maintaining a localized version of "TuxDucks". The development lead, Tux, has made his archive, tux@penguin.org--2004 available at http://tux.org/2004tuxarchive.

As with always, the first thing you should do is to make sure that your arch user id is set. Your arch user id is used to identify changesets that are committed by you. Setting your arch userid is just as simple as telling arch what your email address is:

```
$ baz my-id "John Doe <john@isp.com>"
```

Now, we need to tell arch where Tux's archive is. We perform this with the register-archive command:

```
$ baz register-archive http://tux.org/tux@penguin.org--2004
Registered archive tux@penguin.org--2004
```

If we like, we can take a quick look at Tux's archive, to see what's in it:

```
$ baz rbrowse tux@penguin.org--2004/
tux@penguin.org--2004
  tuxducks
    tuxducks--mainline
      tuxducks--mainline--1
        base-0 .. patch-48
```

In more established archives, often you'll see a lot of categories, branches and versions. In this case though, Tux likes things simple, and he only has one version in his archive, "tuxducks--mainline--1".

However, before we go any further, lets set up our own archive, so that we have a place where we can commit our patches, and make it our default archive:

```
$ baz register-archive john@isp.com--tuxducks /home/john/tuxducksarchive
$ baz my-default-archive john@isp.com--tuxducks
```

Now that we have our own archive. If we want to make a branch of tuxducks, then we can use the branch command in order to make our own branch (which we can commit to, merge into, etc)

```
$ baz get tux@penguin.org--2004/tuxducks--mainine tuxducks
$ cd tuxducks
$ baz branch john@isp.com--tuxducks--1
$ baz commit -s"Starting up my local branch of tuxducks"
```

This works great for this working copy, but what if we delete this working copy and want another one (or perhaps we want two working copies). That's no problem. We can get a new working copy by running:

```
$ baz get tuxducks--mine--1 tuxducks
```

Sure enough, baz went ahead and got you a new working copy (in cvs called a "checkout") and placed it into the tuxducks dir. Now, you can edit the code, commit changes, so forth and so on. Its as if you have your very own secret fork of tuxducks!

But here's a thought; while we're working on our own version of Tuxducks, I'm I'm willing to bet that Tux is going to keep working on his version. If Tux keeps working, he's going to keep commiting new changesets. So how do you get those new changesets?

I'm glad you asked. The process is actually very simple. Just one word of advice: Make sure that your working copy is "clean". A "clean" working copy is one that hasn't changed at all since you originally performed the "baz get" command. If you can't remember whether or not you've changed your working copy since the last time you committed, then you can check for changes by running the following command:

```
$ baz status
```

If you have changes, you'll see a list of files that have been modified, added, so forth and so on. So before you pull in Tux's new changes, you'll want to either commit your changes, do a "baz get" into a different dir and work with that one for the moment, or run the "baz undo" command. For the purposes of this tutorial, I'm going to assume that you don't have any changes.

To actually merge in Tux's new changes, run the following command:

```
$ baz update tux@penguin.org--2004/tuxducks--mainline--1
```

Depending on how much Tux has worked, you may see only a couple changes, or you may see a lot of changes. If you want to see how how he changed his code, try out the nifty changes command with the diff option:

```
$ baz diff | less
```

Now, there is only one thing left to do. Assuming that you are happy with the merge that happened, go ahead and commit your code:

```
$ baz commit -s"Merged in Tux' latest changes"
```

And that's really all there is to it! Now you are able to branch off of anybody else's project, maintain your own changes to your branch of his project, and merge in new changes from the old project. You've done a pretty good job today. :)