Forking upstream packages

Get the sources for your software

The running example will be the custom X11 keyboard layout for the Slimbook SPPP keyboard. Do a little digging and you will find the package that provides the software you want to customize (`dpkg -S` is your friend)

```
apt-get source xkb-data
```

(There's no need for sudo) In our case, this redirects us to xkeyboard-config. If it's not already listed in the software repository, you have to hunt for the software using your favorite search engine. However you get it, the rest of this documentation assumes you are in the root directory of the software repository

debain subdirectory

debain/compat

This file sets the debhelper compatibility version. `debhelper` is a set of simple tools to help you build a Debian package. Level 8 should do nicely. Simplest way to set this:

```
echo 8 > debian/compat
```

debian/source/format

This file describes how the source should be formatted (if the file does not exist, 1.0 is assumed). The idea is essentially to manage patched changes to the code - "native" means roll all the patches together, other schemes organize the patches in different ways.

- "1.0" (or absent): source should be packed either in single .tar.gz (native) or a dual .orig.tar.gz and .diff.gz
- "2.0": ignore, replaced by 3.0 (quilt)
- "3.0 (quilt)": organizes the patches in debian/patches using a file in that directory called series
- "3.0 (native)": pack and unpack everything as-is (like native variant of 1.0) and support multiple compression schemes.

We will almost always prefer using "3.0 (native)". If the source format is anything else, apply the patches (all together in a single commit, or one at a time with a commit each, depending on your preference), If there is a quilt series file, you can move `debian/patches` out of the way, add everything in the directory to a git repo (including `debian` subdirectory, sans `debian/patches`). and make a commit of the original state

```
mv debian/patches /tmp/patches

git init .

git commit -m "Initial commit"

mv /tmp/patches ./patches
```

Then apply all the patches:

```
quilt push -a
```

Or do it one at a time.

```
quilt push 1

# commit

# repeat
```

Either way, make your commits appropriately. When you're finished, be sure to move the folder .pc out of the root of the software repo, because this will indicate to `dpkg` that there are patches to unapply (which of course there no longer are, nor do you want them to be unapplied).

And now you're ready to build the software!

```
dpkg-buildpackage -uc -us
```

Make fixes accordingly.

Aside: Resting software directory after building package (failed or otherwise)

For best results, do the following after every build (once you are finished with the by-products, assuming you have committed or stashed any work you want to save before you built)

```
git reset --hard HEAD

git clean -df
```

That is "do a hard reset of the git repository to the HEAD commit, and force clean (recursively following directories) anything that doesn't belong." Though this won't go so well for any temporary files you might have lying around...

Update debian/changelog

The last step in creating a custom piece of software is changing the revision information. This will be how the Ubuntu software repository on slimbook.labs will know to prefer this package to another package by the same name living in the Maverick old-releases mirror.

A clean way to do this is using `dch` (found in package `devscripts`). Assuming the current software has the following version format

```
${upstream_revision}-${debian_revision_num}${debian_revision_rest}
```

Keep `${upstream_revision}`, increment `${debian_revision_num}`, and replace `${debian_revision_rest}` with `genesi0` . So if the revision was `1.8-1ubuntu.1~10.10.1` you will want to update this to `1.8-2genesi0` with this command:

```
dch -v 1.8-2genesi0
```

This will open up a text editor (or ask you to select one). Add your comment (and make sure your name / email information is correct) then build again. If successful, you are ready to test your manual build by installing the resulting `.deb` files using `dpkg`

Any following, major (stable) changes or milestones to the code should increment the number after `genesi`.