Proposed design to the Bioc Build System

Hervé Pagès

last update: 10-21-2005

I. INTRODUCTION

The BBS (BioC Build System) consists of 2 subsystems: the *Build System* (running on several *build nodes*) and the *Publishing System* (running on the *publishing node*).
The *src pkgs* (source packages) and *bin pkgs* (binary packages) produced by the *build nodes* are called *build products*. All other files and directories produced by the *build nodes* are called *build results*.
Names of *build products* and *build results* are represented in **bold**.
During the builds, all *build products* and *build results* for a given BioC version (branch) are placed in a central location referred to as the BBS_LATEST_BASEURL. For the 1.8 builds, BBS_LATEST_BASEURL is something like http://gopher5/.../1.8/latest (directory).
Depending on its role in the Build System, a *build node* might need write access here (using mv, rsync of ftp). All *nodes* (*build nodes* + *publishing node*) must be able to retrieve files from there via HTTP.

II. MAIN TASKS

1) The *Build System* does the following:

The "src build" task

On each *build node*, **build** the BioC *src pkgs* from the *pkg dirs*:

- COMMAND: Run 'R CMD build <*pkg dir*>' on each *pkg dir*.
- BUILD PRODUCTS (1 file per *pkg dir*): **<pkg>_x.y.z.tar.gz** (the *src pkg*).

- BUILD RESULTS (3 files per *pkg dir*): **<pkg>.src.out**, **<pkg>.src.err** and **<pkg>.src.report**.

- Put the above products and results under BBS_LATEST_BASEURL/<*build node*>/src/

The "src check" task

On each *build node*, **check** the BioC *src pkgs* produced during the "src build" task:

- COMMAND: Run 'R CMD check <*src pkg*>' on each *src pkg*.

- BUILD PRODUCTS: None.

- BUILD RESULTS (1 dir + 3 files per *src pkg*): **<pkg>.Rcheck/**, **<pkg>.check.out**, **<pkg>.check.err** and **<pkg>.check.report**.

- Put the above results under BBS_LATEST_BASEURL/<*build node*>/

The "bin build" task

On some chosen *build nodes* only, **build** the BioC *bin pkgs*:

- COMMAND: Run 'R CMD INSTALL –build <*src pkg*>' on each *src pkg*.

- BUILD PRODUCTS (1 file per src *dir*): **<pkg>_x.y.z<platform suffix>** (the *bin pkg*).

- BUILD RESULTS (3 files per *src pkg*): **<pkg>.bin.out**, **<pkg>.bin.err** and **<pkg>.bin.report**.

- Put the above products and results under BBS_LATEST_BASEURL/<*build node*>/bin/

NB: *Bin pkgs* have a platform specific suffix: ".zip" on Windows, ".tgz" on OS X, "_R_x86_64-unknown-linux-gnu.tar.gz" on x86_64 Linux, etc...

2) The *Publishing System* does the following:

The "publishing" task

Publish the following stuff to the Bioconductor website:

- Publish all *build results* produced by the *Build System* to the "nightly build" section of the website.

- Publish *src pkgs* (from one chosen *build node* only) + *bin pkgs* (from one Windows *build node* only) to the repos section of the website. Replace a previously published pkg by the new one only if the new one was build with no errors and has a higher version string.

- Publish all the vignettes extracted from the *src pkgs* (from one chosen *build node* only) somewhere on the website so that people can download them without having to download the corresponding *src pkgs*.

III. *NODE* ROLES

A *build node* can have several roles, depending on what we want it to do:

- It can be "*the* prelim repos builder" (see STEP 1 below). Only one *build node* needs to perform this step.

- It can be "*a* src builder" (see STEPS 2,3 below) i.e. *a* machine that builds *src pkgs*. In theory 2 *src pkgs* build from the same *pkg dir* on 2 different *build nodes* should be (almost) identical, except for timestamps and (on very rare cases) for vignettes (due to obscure platform related issues).

- It can be "*a* src checker" (see STEPS 2,3,4 below) i.e. *a* machine that checks s*rc pkgs*.

- It can be "*a* bin builder" (see STEPS 2,3,5 below) i.e. *a* machine that builds the *bin pkgs* from the *src pkgs*.

The *publishing node* is the node that performs the "publishing" steps (see STEPS 6,7,8 below).

IV. LOCAL LAYOUT

On each *build node*, a given BioC branch (e.g. 1.8) is build under its own directory called the BBS_WORK_TOPDIR. For efficiency, the BBS_WORK_TOPDIR should be **local** to the *build node* (not NFS mounted).

On "*the* prelim repos builder", the BBS_WORK_TOPDIR layout looks like this:

```
  BBS_WORK_TOPDIR/
    madman-wc/
    madman/
```

On "*a* src checker" or "*a* bin builder", it looks like this:

```
  BBS_WORK_TOPDIR/
    madman/
```

No BBS_WORK_TOPDIR is needed on the *publishing node*.

V. 8 STEPS

It's convenient to define the 8 following steps (elementary tasks) that we can put together in order to achieve the 4 main tasks described at the beginning:

STEPS 1 and 2 are preliminary steps. STEP 1 is a global preliminary step: it needs to be performed on one *build node* only ("the prelim repos builder") before any other *build node* can start to do anything. STEP 2 is a local preliminary step: it must be run on each *build node* before anything else (after STEP 1 has successfully terminated on the "prelim repos builder").

STEPS 3,4,5 correspond to the "src build", "src check" and "bin build" tasks.

STEPS 6,7,8 are the "publishing" steps (performed on one the *publishing node* only).

STEP 1: Make "madman.tgz" and "latest prelim repos"

Do a "rolling backup" of BBS_LATEST_BASEURL

```
cd BBS_WORK_TOPDIR
rm -rf madman madman.tgz

svn up madman-wc
svn export madman-wc madman
tar zcf madman.tgz madman
mv madman.tgz BBS_LATEST_BASEURL/

mkdir BBS_LATEST_BASEURL/src
mkdir BBS_LATEST_BASEURL/src/contrib

cd madman/Rpacks
Get list of pkgs from manifest file
for each pkg in list do:
  R CMD build --no-vignettes <pkg>
  mv <pkg>_x.y.z.tar.gz BBS_LATEST_BASEURL/src/contrib/
Make the "latest prelim repos" control file
```

STEP 2: Install all pkgs from "latest prelim repos"

-- no details for now --

STEP 3: Build the *src pkgs*

```
mkdir BBS_LATEST_BASEURL/<build node>

cd BBS_WORK_TOPDIR
rm -rf madman madman.tgz
wget -N BBS_LATEST_BASEURL/madman.tgz
tar zxf madman.tgz

cd madman/Rpacks
Get list of pkgs from "latest prelim repos"
for each pkg in list do:
  R CMD build <pkg> > <pkg>.src.out 2> <pkg>.src.err
  Make the <pkg>.src.report file
  cp <pkg>_x.y.z.tar.gz BBS_LATEST_BASEURL/<build node>
  mv <pkg>.src.out <pkg>.src.err \
     <pkg>.src.report BBS_LATEST_BASEURL/<build node>/
```

STEP 4: Check the *src pkgs*

```
cd BBS_WORK_TOPDIR/madman/Rpacks
Get list of src pkgs in current dir
for each pkg in list do:
  R CMD check <pkg>_x.y.z.tar.gz \
    > <pkg>.check.out 2> <pkg>.check.err
  Make the <pkg>.check.report file
  mv <pkg>.Rcheck <pkg>.check.out <pkg>.check.err \
     <pkg>.check.report BBS_LATEST_BASEURL/<build node>/
```

STEP 5: Build the *bin pkgs*

```
cd BBS_WORK_TOPDIR/madman/Rpacks
Get list of src pkgs in current dir
for each pkg in list do:
  R CMD INSTALL --build <pkg>_x.y.z.tar.gz \
    > <pkg>.bin.out 2> <pkg>.bin.err
  Make the <pkg>.bin.report file
  mv <pkg>_x.y.z<platform suffix> \
     <pkg>.bin.out <pkg>.bin.err <pkg>.bin.report \
     BBS_LATEST_BASEURL/<build node>/
```

STEP 6: Publish the "src build", "src check" and "bin build" results to the Bioconductor website.

-- no details for now --

STEP 7: Publish the src pkgs and bin pkgs to a public repos (update only).

-- no details for now --

STEP 8: Publish the vignettes to the Bioconductor website.

-- no details for now --

VI. SCHEDULING THE *NODES*

Here is the list of nodes we use and the roles they have:

	platform	***prelim repos builder***	***src builder***	***src checker***	***bin builder***	***publisher***
gopher5	Linux x86_64	1	2 3	4		6 7 8
wellington	Linux i686		2* 3	4		
churchill	sparc		2* 3	4		
lemming	Windows 2003 Server		2* 3	4	5	
walpople	Windows XP		2 3	4	5	

We could also setup a "bin builder" node only (no checks) like this:

```
binbuilderonly: 2* 3  5
```

On each node a step can only be started if the preceding step terminated successfully. A node might also have to wait that some step terminates on **another** node before it can start to perform a step e.g. 2* here means that STEP 2 can only be started if STEP 1 terminated successfully on gopher5.

It would be nice to implement a cross machine locking/logging mechanism. Not the first priority though...