

## Outline of tutorial

- [1] Overview of software system used by MIPP
- [2] Getting connected: cvs
- [3] Getting, building, and running code: SRT
- [4] Building a simple analysis module and job
- [5] Details of some existing packages

I hoping you will be able to "type along" with me. This will require that you have access to a site with a recent version of the software installed.

If you don't you can use my set up at IU. (Instructions to follow...)

# *Overview of MIPP software*

Believe it or not, this is covered in the OfflineUsersManual!

<http://enrico1.physics.indiana.edu/mipp/OfflineUsersManual/>

[main](#) for HTML version

or

[OfflineUsersManual.ps](#) for postscript version. I'll pick up from the 1<sup>st</sup> chapter there for the overview.

# Getting Access

Current (and all past) versions of the MIPP code are stored on a CVS served at Fermilab ([cdcv.s.fnl.gov](http://cdcv.s.fnl.gov)). To connect you need to set the following:

```
setenv CVSROOT e907cvs@cdcv.s.fnl.gov:/cvs/e907
```

To check the code out from there you need permissions to connect either via kerberos or via ssh.

# *CVS Access via kerberos*

[1] At FNAL, kerberos is probably the way to go. Off the FNAL site, its a little bit trickier...

[2] You will need to be added to the .k5login on the server side by one of the experts e-mail: [soltz@llnl.gov](mailto:soltz@llnl.gov), [raja@fnal.gov](mailto:raja@fnal.gov), or [messier@indiana.edu](mailto:messier@indiana.edu)

[3] Next, you need to make sure CVS uses a kerberized version of rsh or ssh:

```
% which rsh  
/usr/kerberos/bin/rsh
```

(Could also be [/usr/krb5/bin/rsh](#))

[4] Get a kerberos ticket:

```
% kinit
```

You should now be OK to connect to the cdcvs server

# CVS Access via ssh [1]

[1] Off the FNAL site ssh is probably the easiest way to get access

[2] For this you'll need an ssh-1 public key:

```
% ssh-keygen -t rsa1
```

```
...
```

```
% cd ~/.ssh
```

```
% ls
```

```
identity identity.pub known_hosts
```

Your pass phrase is like a password but should **not** be your system or kerberos password!! You can also have ssh-2 style keys. If you use the same password for those all types will work the same.

[3] The server needs to have your [identity.pub](#) installed. Forward that file (not the identity file which you should **never** give out!!) to one of the experts ([messier@indiana.edu](mailto:messier@indiana.edu), [raja@fnal.gov](mailto:raja@fnal.gov), [soltz@llnl.gov](mailto:soltz@llnl.gov))

## *Access via ssh [2]*

[4] Once your ssh key is installed, you will need to set your session up to make the connection using your key. Put these two lines into your .login file:

```
`eval ssh-agent -c`  
ssh-add
```

(switch “-c” to “-s” if you're not a C-shell user)

[5] Add this line to your .logout file:

```
ssh-agent -k
```

[6] These files will get sourced every time you login and out. For this first this-time-only session source the .login by hand:

```
% source .login
```

# *Taking CVS out for a spin*

OK we should all be able to connect now. For example, using SSH:

```
% env | grep CVS
CVSROOT=e907cvs@cdcvs.fnal.gov:/cvs/e907
SRT_SAVE_CVSROOT=
CVS_RSH=ssh
% env | grep SSH
SSH_CLIENT=131.225.56.118 32780 22
SSH_CONNECTION=131.225.56.118 32780 129.79.160.239 22
SSH_TTY=/dev/pts/5
SSH_AUTH_SOCK=/tmp/ssh-dPpF1694/agent.1694
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
% cvs get CVSROOT
```

Only this last one is needed. It will checkout a copy of the CVSROOT directory from the repository

# What's in CVSROOT?

CVSROOT contains administrative files for the repository. In general non-experts will never want to mess with the files in it. A few exceptions are:

**\*.list** – These are lists of e-mail addresses of people who get notified when changes are made to files in the repository

**modules** – These are a list of short hand names for directories in the repository. If you author a new package you'll want to add it to the list in the modules file.

Let's add ourselves to the **offline.list** file

```
% cvs update -d  
% [edit offline.list adding your e-mail]  
% setenv EDITOR 'emacs -nw'  
% cvs commit
```

cvs uses the EDITOR to allow you to make a log file comment about your modification. Emacs is just an example...

# *You can trust CVS!*

CVS is very good about merging changes to different parts of the same file. In general, updating, and committing often is your best bet to stay in sync. with the latest code! If you goof up it is very easy to roll back...

Other useful cvs commands:

`cvs diff` – differences between local version and server

`cvs log` – summary of recent changes to code

# *Software Release Tools (SRT)*

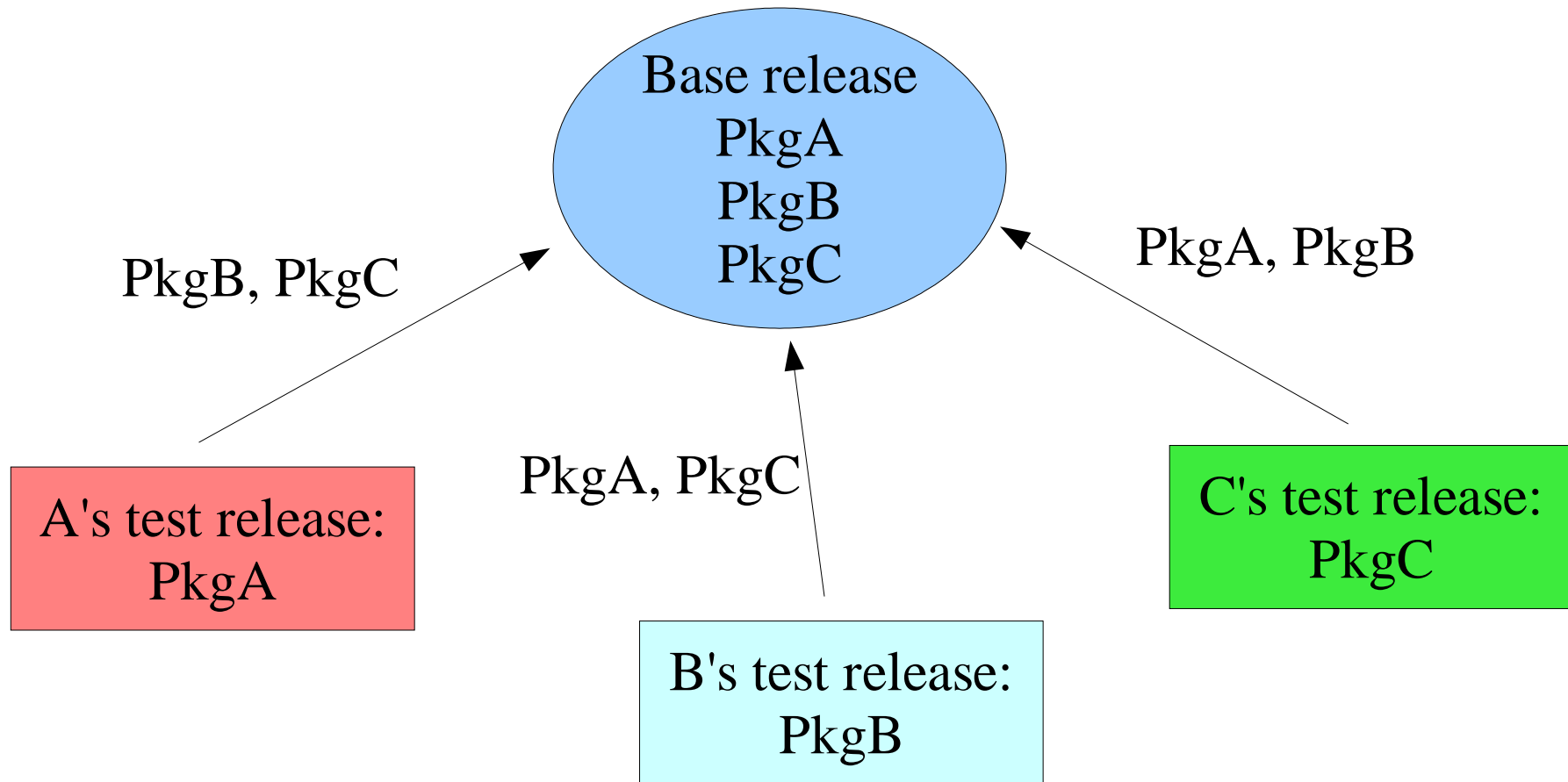
MIPP uses FNAL's Software Release Tools to manage software packages

SRT manages:

[1] “releases” of code (collections of tagged package versions)

[2] Build system (ie. GNUmakefiles)

# *Base releases, test releases and all that...*



Several users at a site reference a single “base” release of the code maintained by the site librarian

Users develop their own code in local “test” releases. The test releases reference the base release for any code the user is not working on

# *SRT base release*

To work at a site you need to know where the base release(s) live

To set up a base release, do this (for example) in your `.cshrc` file:

```
source /usr/local/mipp/srt/srt.csh
if ($#argv == 1) then
  set release = $argv[1]
else if ($#argv == 0) then
  set release = development
endif
srt_setup SRT_BASE_RELEASE=$release
```

This sets up the “development” release by default. If you're actively working on code “development” is probably what you want. If you're just running code, you may want to use a tagged release (R03.06.05 eg.)

In this example, the base release was located in `/usr/local/mipp`

# *SRT base release*

You should now have several things in your environment

```
% env | grep SRT
```

For example, the event display:

```
% rehash
```

```
% which evd
```

```
/usr/local/mipp/releases/development/bin/Linux2-GCC/evd
```

This shows that I'm picking up the event display from the development base release (/usr/local/mipp/development)

Download this file:

<http://enrico1.physics.indiana.edu/mipp/data/pp120gev.root>

and take a look:

```
% evd pp120gev.root
```

# *SRT test release*

Now, let's try to make a test release using the development as a base release. The command for that is called “newrel”

```
% mkdir mippsoft  
% cd mippsoft  
% newrel -h  
% newrel -t development mytest  
% cd mytest  
% ls  
bin doc GNUmakefile include lib man results tmp
```

Notice that this looks just like an official release. You've got a place for libraries (lib) and binaries (bin) and all that. Only difference you don't have any packages yet...

# *SRT test release*

To inform SRT that you are working with “mytest” as a test release you need this command:

```
% pwd  
/home/janedoe/mippsoft/mytest  
% srt_setup -a
```

The “-a” says “set up the test release in the current directory”

Now you'll notice that:

```
% echo $SRT_PUBLIC_CONTEXT  
/usr/local/mipp/releases/development  
% echo $SRT_PRIVATE_CONTEXT/  
/home/janedoe/mippsoft/mytest/
```

Point to the “PUBLIC” (ie. base) release and the “PRIVATE” ie. test release

# *SRT test release: adding packages*

Ok, now, let's populate our release with an existing package. For example, suppose we wanted to make some changes to the RICHReco package:

```
% pwd
/home/janedoe/mippsoft/mytest
% addpkg -h RICHReco
Adding package "RICHReco" to ".".
cvs checkout -P RICHReco
Adding package "RICHReco" to ".".
cvs checkout -P RICHReco
```

Notice that SRT does the cvs checkout for you. Now if you do an “ls” you should see the RICHReco package in your release.

## *SRT test release: fixing the include links...*

SRT puts include files from your test release first in the search path. ie. to your test releases include/Package directory first. However, it often messes up the include link. To check do this:

```
% cd include  
% ls -l
```

You should see that RICHReco is a link to ../RICHReco. If its not, fix it by hand:

```
% rm RICHReco  
% ln -s ../RICHReco
```

This may get fixed in the future, but for now that's the fix. Let's go back to the top of our test release before going on:

```
% cd ..
```

# *Building*

To build the packages in the release type:

```
% gmake
```

SRT will rebuild all packages that have been modified as well as the packages that depend on them. Just to check, this should show you the RICHReco library:

```
% ls lib/*
```

You can also build just one package:

```
% cd RICHReco
```

```
% gmake
```

# *Building*

Other useful gmake commands:

gmake lib - build the library

gmake bin - build the binaries

gmake tbin – build any test binaries

gmake clean – clean the package/release build

These can also be done package-by-package from the top of an SRT release:

gmake Package.clean

gmake Package.lib

gmake Package.bin

gmake Package.tbin

# *SRT Makefiles*

One nice thing SRT does for you is simplify what goes into your makefiles. Take a look at the RICHReco GNUmakefile for the basic pattern. You can probably just copy this for any new package you start. It assumes the following conventions which MIPP uses:

- [1] All source files to be compiled into libraries end in .cxx
- [2] Source files which contain main() end in .cc
- [3] All include file names end in .h
- [4] Test programs live in the “test” subdirectory

You'll want to take a look at test/GNUmakefile also to see how things go there.

# *Removing a package*

After you've made your changes, tested that they work and have committed the code, you can remove the package from your test release:

```
% cd $SRT_PRIVATE_CONTEXT  
% gmake RICHReco.clean  
% rmpkg RICHReco
```

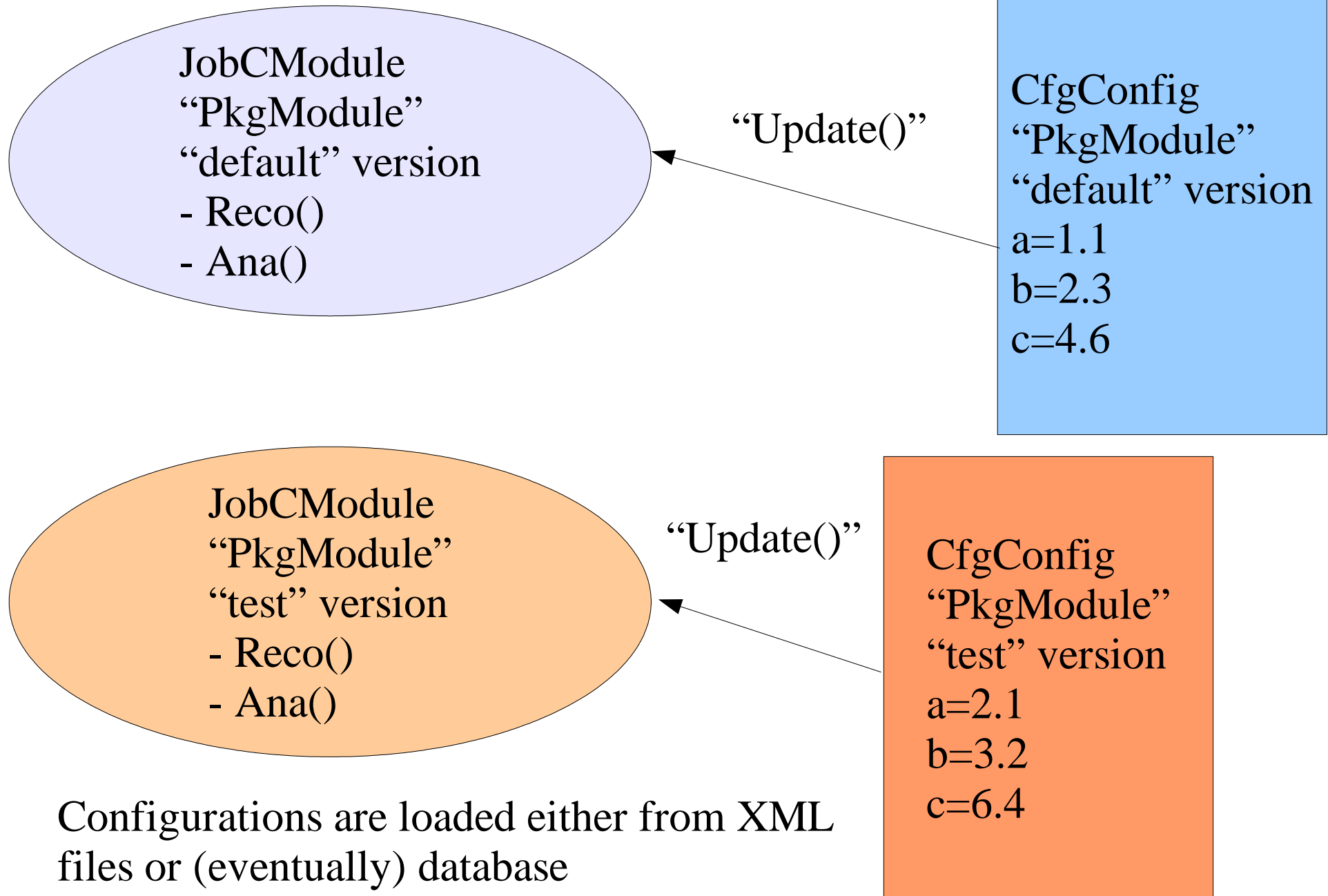
# *Analysis modules*

The JobControl package defines the basic unit for user reconstruction and analysis: [JobCModule](#)

Users sub-class JobCModule and then can plug their code into the analysis program “[anamipp](#)”

Module configuration data is held by the [CfgConfig](#) class. Updates are handled by the [Config](#) package

# Analysis modules



# *Specific Example: RICHReco*

Eventually I hope to put an example module “Examples” together

Until then let's use the RICHReco package and the RICHCircFit module as examples.

Here I'll add lib using screen shots of the code...

Basic command is:

```
anamipp -x circfit.xml pp120gev.root
```

`circfit.xml` : Job description

`pp120gev.root` : input root file

# *Where to find data?*

MC data: I have the one file at the moment. Working on generating others using e907mc

“Real” data:

e907daq or mon: /data/1 or /data/0 \*.raw

To convert raw to root format use “raw2root” utility:

```
% raw2root -o output.root input.raw
```