CLAS-NOTE 2004-006
CVS: docs/offline/offline_faq.tex

CLAS Offline Frequently-Asked Questions

Mark M. Ito
February 18, 2004

Contents

1 CLAS Software

1.1 Where do I find general information about CLAS software?

The main page that documents our software is at http://clasweb.jlab.org/offline/. For example this FAQ is linked directly from that page.

1.2 Where do I get the document describing the code management system?

Go to http://clasweb.jlab.org/offline/cms/ for an HTML version,
or http://clasweb.jlab.org/offline/cms.ps for a postscript version.

1.3 How do I find out what the current production release is? The current development release?

Do the following directory listing:

```
ls -l $BUILDS

lrwxrwxrwx   1 claslib  clas     12 Mar 26 02:02 DEVELOPMENT -> release-1-15/
lrwxrwxrwx   1 claslib  clas     12 Mar 26 02:02 PRODUCTION -> release-1-14/
drwxr-sr-x   6 claslib  clas   4096 Feb 18 14:22 release-1-11/
drwxr-sr-x   6 claslib  clas   4096 Feb 12 15:54 release-1-12/
drwxr-sr-x   6 claslib  clas   4096 Feb 24 09:42 release-1-13/
drwxr-xr-x   6 claslib  clas   4096 Mar 19 16:14 release-1-14/
drwxr-xr-x   6 claslib  clas   4096 Mar 25 18:16 release-1-15/
```

This tells us your that release-1-14 is production and release-1-15 is development.

1.4 How do I get more information on the releases?

The release notes are cataloged at http://clasweb.jlab.org/offline/release_notes/tags.html.

1.5 How do I get the CLAS environment set up at JLab?

I recommend the following:

In your .cshrc:

```
alias setup_clas \
    source /group/clas/builds/PRODUCTION/packages/cms/jlab.cshrc
```

and in your .login:

```
setup_clas
```

The advantage of not having `jlab.cshrc` sourced in your `.cshrc` (notwithstanding the name) is that child processes will inherit the CLAS environment definitions from their parents. If you include the sourcing of `jlab.cshrc` in your .cshrc it will wipe out any parental definitions.

1.6 I'm having a make problem and I need to see the values of the makefile's internal variables. How do I do this?

If you are using an Arne-style makefile, type

```
make help
```

You will get a long list of the variables that are defined.

Your variable not listed? Check out the packages/cms module and add printout of the variable in the help target of packages/cms/Makefile. It is obvious how to do this. To use the new upgraded Makefile, type

```
setenv CLAS_CMS <name of your cms directory>
```

and please check in your change.

1.7 How is the $CLAS_PARMS directory managed?

- The area is backed up by the Computer Center nightly. It is also .snapshot'ed.
- There is no fixed Unix privilege protection on the files.
- To reduce confusion, you should announce any changes you make to files in $CLAS_PARMS to clas_offline@jlab.org.
- The Maps directory in $CLAS_PARMS is a nightly dump of the latest version of the main CalDB run index (with constants included, of course). See FAQ-1.8 for more information.
- Proposals for a real management scheme are welcome.

1.8 How is it that the Mapmanager files in $CLAS_PARMS/Maps have recent versions of constants even though they are supposed to be frozen?

At midnight every night, a cron job runs that regenerates the Mapmanager files from the current constants kept in the CalDB. The script that is run is

```
/group/clas/tools/caldb/map_refresh.sh /group/clas clasdb -1000000 1000000
```

which means that all runs from -1 M to +1 M are copied from the clasdb.jlab.org database server, and deposited in the parms directory found under /group/clas.

The old copies of the map files are kept in /group/clas/parms/Maps_old. The frozen version (frozen before copying to the CalDB) is kept in /group/clas/parms_frozen/Maps.

One *caveat*: the RUN_CONTROL map file is simply copied from version to version. It is not regerated from the CalDB. The offline copy is kept up-to-date by automatic copies coming from the online system.

2 Disk Space

2.1 What are the various types of public disk space available at JLab for CLAS?

As of September 24, 2002:

Type	Size (TB)	Availability	User Writable
scratch	0.56	Lab-wide	yes
work	5.01	CLAS-only	yes
CLAS general cache	2.20	CLAS-only	no
CLAS DST cache	2.10	CLAS-only	no
default general cache	0.09	Lab-wide	no
farm staging general cache	2.35	Lab-wide	no

2.2 How is space on the scratch disk managed?

Every night a program is run that examines the files on the scratch disk. Files that have not been accessed within the last 30 days are deleted. The JLCC maintains this system.

2.3 What is the work disk for? How is it managed?

Work disk is used for temporary storage of large volumes of data. Any CLAS user (as defined by membership in the "clas" Unix group) has write privilege on these partitions. There are no disk space quotas on the work disk.

We have two classes of work disk, distinguished by the method for managing space.

Auto-managed work partitions are cleaned by a script every night to create free space. The script uses the following algorithm on a partition-by-partition basis.

1. If the amount of free space on the partition is less than 30%, an attempt is made to delete files until that amount is free.
2. If a file has been accessed within the last 10 days, then it is exempt from deletion.
3. Files are deleted in access time order, oldest files first.
4. Directories that have remained empty for more than a week are removed.

Note that because of the exemption for newly-accessed files, it is possible that the script will not reach its target for free space.

There are seven auto-managed work disks: `/work/clas/disk1` through `/work/clas/disk7`.

Hand-managed partitions are administered by particular individuals who adjudicate conflicting demands for space. They are meant for files that we do not want to subject to possible auto-deletion. Among the hand managed partititions there are two types:

1. Special purpose hand-managed partitions

   Each partition is ear-marked for a particular task. If you have a project that requires non-volatile disk space for one of these purposes, contact the appropriate administrator.

Partition	Administrator	Purpose
`/work/clas/calib`	Franz Klein	Calibration tasks
`/work/clas/gsim`	Will Brooks	Simulation tasks
`/work/clas/physana.old`	Stepan Stepanyan	Physics analysis tasks
`/work/clas/production`	Mark Ito	Large-scale data processing

2. General purpose hand-managed partitions (`pdisks`)

   These partitions are meant for general purpose analysis work and are available to any collaborator who needs space that is not subject to auto-deletion. See FAQ-2.4, 2.5 and 2.6 for more details.

2.4 How are the general purpose hand-managed work disks (pdisks) managed?

Each partition has a single manager. The term of the manager is at most six months. The manager may retire earlier than that if he or she so chooses. The manager makes all decisions regarding who may use the partition, how much space they may use and most importantly, what files should be deleted if/when more space on that partition is needed. Managers are volunteers. If there are more volunteers than partitions, then we a waiting list is formed, first come, first served. No one may manage more than one partition. If we have partitions with no managers, they are run as regular auto-managed work disks.

When a manager takes over a partition, then he or she may, at his or her discretion, at the time of his or her choosing, delete any and all files on that partition. If file protections make this difficult, we ask the Computer Center to help.

2.5 How do I get space on the pdisks?

If you need quasi-permanent work space, you can (a) volunteer to be a work space manager and gain control of an entire partition by sending email to Mark Ito (`marki@jlab.org`) or (b) ask one of the current managers if you can have some permanent space on his or her partition. The disks are not designated for any particular analysis topic. Users doing different things can co-exist on the same partition, as they do on the auto-managed work disk.

Remember, work disk is meant for temporary storage only. It is not backed up. This work disk is not different in that regard. If you cannot risk losing data on any work disk due to hardware failure, it should be written to the silo. If data exists on the silo, and is needed permanently on disk, it should be put on the DST cache partitions and not on the work disks.

2.6 Who are the pdisk managers?

Find the list at

`http://clasweb.jlab.org/offline/pdisk_managers.html`

The list is also linked from the CLAS/JLab Computing Resource Status page at

`http://clasweb.jlab.org/offline/resource_status.html`

2.7 How do I tell how much space is being used on the various work disks (both auto-managed and hand-managed)? How much of it has been used recently?

See the Computer Resources page at

`clasweb.jlab.org/offline/resource_status.html`

Click on "Work Disk Usage".

2.8 How do I tell how much space I am using on the auto-managed work disks? How much others are using?

See the Computer Resources page at

`clasweb.jlab.org/offline/resource_status.html`

Click on "Auto-Managed Work Disk Usage".

2.9 I am extending the life of my files on the auto-managed work disk by manipulating their access dates periodically. That way the auto-deleter does not get them. Is that OK?

No.

What you are doing is effectively reserving a portion of the work disk for yourself, for an indefinite period of time, without consultation with your collaborators who must share the same disk resources. If you do this with a cron job, then you are doing so quasi-permanently.

The auto-deletion algorithm is laughably easy to defeat. We could make the algorithm smarter, but then we run the risk of entering into an escalating algorithm war with clever collaborators. We are on the honor system here.

There are such things as emergencies. If need to leave town for your great aunt's funeral and you touch all your files before you leave, that's fine. No one will object. What is objectionable is touching files for weeks on end.

If the data is important enough to keep on disk for several weeks, they are probably important enough to write the silo. Once on the silo, they can be recovered if the auto-deletion gets them, and it becomes possible to store them on one of the cache disks as well.

If you need quasi-permanent storage of large number of files, please use the DST cache. See FAQ-2.20.

2.10 What if I manipulate the access dates anyway?

Our policy is that after an email warning, if the practice continues for another week, you will lose access to the farm and ifarm nodes. You will then have to apply to the powers that be for reinstatement.

2.11 I checked out some files from CVS on the work disk yesterday, and a lot of them were deleted last night. How come?

In a CVS checkout, files are created on the disk with a creation and access date of the version fetched from the repository. This means that in most cases, the checked-out files will be much older than a week and will be at the top of the list of files subject to deletion.

The first comment is that unless you are doing some kind of temporary test of the existing versions, the work disk is a spectaculary bad place to check out the code. (a) It is not backed up. (b) It's subject to auto-deletion. A better choice would be to use your home area for the source code, and build only the libraries and executables on the work disk (by setting your `$TOP_DIR` appropriately). The source code does not take up much space and if the stuff on the work disk vanishes, it can be re-made.

Second, if you really must build on the work disks, you might as well go ahead and update the access dates of all the source files. This sin will be forgiven: the fact that the files are in a CVS-type tree will make the motivation clear and, again, the source files don't take up that much space.

2.12 What is `/work/clas/farm_output` for?

The `/work/clas/farm_output` is intended for use by farm jobs as temporary storage space for files that will be written to the tape silo. This practice makes writing to the silo more efficient by writing large amounts of data with each tape mount.

The function of accumulating disk files before shipment to the silo could be provided by our standard work disks (`/work/clas/disk1`, *etc.*). On the other hand, large volumes of new data on those disks will displace older pre-existing files as the auto-deletion algorithm does its work (see FAQ-2.3). If this new data is only to be stored on a temporary basis (*i. e.*, only until written to tape), then we run the risk of ending up with the standard work disks being full of only unneeded files. To address this pre-silo accumulation function, which is critical during Pass 1 processing, we have `/work/clas/farm_output` which is ear-marked for this purpose.

Because of its function, the auto-deletion parameters for `/work/clas/farm_output` are different from those for standard work disks. The guaranteed file life is three days and the disk is checked every four hours. The goal for free space is 30%. In all other respects the disk is treated the same as our standard work disks, in particular users can write to it directly.

2.13 What is the general cache disk for? How is it managed?

The general cache disks are intended for temporary storage of files that originate from the tape silo. One major use is for staging raw data files for later transfer to the batch farm

for Pass 1 processing. Another example of its use is to retrieve large numbers of HBOOK files for physics analysis, *en masse*, although for the latter purpose DST cache may be more appropriate (see FAQ-2.20).

The cache areas are `/cache/mss/clas` and `/cache/mss/home`. They can only be filled with files from the silo and can only be written to directly by root. User requests for files are submitted with the `jcache` command. For more on `jcache`, see FAQ-2.19.

The directory structure for `/cache/mss` is exactly the same as that on `/mss`. If you know the file name stub name under `/mss`, you get the output file name on the cache disk by prepending `/cache` to that file name. All of the "files" in the `/cache/mss` tree are actually symbolic links. The files themselves are stored on an ensemble of disk partitions separate from the physical `/cache` partition. The "real" files linked from one cache directory often live on more than one physical partition.

The disk is managed by the JLCC (with one twist, see below). If there is no space to fulfill a jcache request, then the oldest files, by access date, are deleted until there is enough space. This means two things: (a) the disk will end up being 100% full all of the time and (b) there is no guaranteed period for file life on the disk.

The advantages of this scheme are:

It's never really full. Even though the partitions may be 100% full of files, there's always room for more files. The jcache command will delete space until it can satisfy its request.

There's no duplication. Since all files come from the silo and are put in an unambiguous location, you cannot have a file appear more than once on the cache disk. The jcache command itself checks to see whether a file is already there. If so, it does nothing.

Files are easy to find. You do not have to search an entire disk partition to determine whether a particular silo file exists on the cache disk. The directory name on the cache is derived from the directory name of the silo stub file so there is only one place it can be.

Useful files are kept. Since age is judged by access time, any read of a file resets its age to zero. Files that are being used often tend to be young and can live on the disk for an indefinite amount of time.

Note that the existence of any file on the disk that is no longer needed has the negative side-effect of pushing older files to deletion ahead of it. To deal with this, there is a user command option, `jcache -d`, that will artificially set the access date of a file to be in the distant past. Files so marked look very old and will be the first to go when space for new files is needed.

Here is the twist: we, independently of the JLCC, enforce quotas on the general cache disk.

2.14 How do the quotas on the general cache disk work?

1. The quota is only enforced when there are no files marked for early deletion. If any such file exists, no quota checking will occur.

2. The quota is applied on a per directory basis, not on a user-by-user basis. For example, `/cache/mss/clas/e1a/data` has a quota and so does `/cache/mss/clas/eg1a/production/pass1/dst`.

3. If a directory is over quota, files in that directory are marked for early deletion, starting with the oldest until the sum of the unmarked files is below the quota. Age is by access time.

4. Quotas are checked every fifteen minutes.

5. The actual value of the per directory quota depends on how long files are living on the cache disk. This is based on the age of the oldest, unmarked file.

Age (days)	**Quota (GB)**
0.0-3.0	200
3.0-7.0	400
> 7.0	no quota

Note that if a directory stays below the quota, then this procedure has no direct effect for that directory.

The intent is to target those users that are caching large amounts of data and not marking them for early deletion when they are done with them, or are not using them promptly after they appear on the disk. These practices reduce the lifetime of all files on the cache disk, making life miserable for all cache disk users.

2.15 How do I check on how much of my general cache disk quota has been used?

See the Computer Resources page at

`clasweb.jlab.org/offline/resource_status.html`

Click on "General Cache Disk Usage".

2.16 How do I find the access age of my files on the general cache disk?

See the Computer Resources page at

`clasweb.jlab.org/offline/resource_status.html`

Click on "General Cache Disk File Listing".

2.17 How do I delete files from the general cache disk?

To delete a file from the general cache use the `-d` switch of the jcache command, for example:

```
jcache -d /mss/clas/e1e/data/clas_036577.A30
```

This does not actually delete the file. Rather it marks it for early deletion. This is done by artificially setting the access date of the file to a time in the distant past. That makes the file look old to the general cache management software and therefore identifies the file as a candidate for deletion whem more space is needed for new files.

2.18 Why would I ever want to delete files from the general cache disk? Won't the disk manager delete them for me eventually?

Yes, eventually the disk manager will delete your files for you, but if you mark your files for early deletion when you are done with them, you extend the life of files that are older than yours. If your files get deleted today, other files may live to see another day. In fact, the files you save may be your own!

2.19 How does the `jcache` command know that it should write my files to the CLAS general cache partitions and not those for Halls A or C?

Actually a rather basic argument to the `jcache` command that we habitually omit is the `-g` <`cache group`> switch. This selects one of several disk cache group areas for storage of the the requested files. If this is omitted, the the command checks for the Unix group affiliation of the user making the request. If the user belongs to the `clas` Unix group, then `-g clas` is automatically assumed.

2.20 What is the DST cache disk for? How is it managed?

The DST (data summary tape) cache is used for quasi-permanent storage of data files. The intended use for this space is for large collections of highly compressed data, thus the name. That having been said, the space can be used for any purpose that requires long-term storage of data where the files are not subject to any automatic deletion scheme.

The DST cache is similar to the general cache in that it can only be filled with files from the tape silo. Unlike traditional cache, once a DST cache area fills up and no space exists to satisfy a write request, the request will fail. In this situation, files must be removed explicitly by the users to make room for new files.

Just like general cache, files are accessed via links under `/cache/mss`. The name of the cache file is the same as the name of the corresponding silo stub file only with `/cache` prepended. The actual disk files are kept in partitions separate from `/cache`. One thing to note about this scheme is that you cannot distinguish between files that are in the general cache and those in DST cache by the directory location of their symbolic links. You have to look at where the links point to.

2.21 Who gets to use the DST cache?

We allocate portions of DST cache to Run Groups. An allocation can consist of a fraction of a physical partition, an entire partition, or several partitions. Right now the standard allocation is about 200 GB. The Run Groups are free to use the space for any purpose they see fit.

2.22 How do I get a DST cache partition for my Run Group?

Send an email to Mark requesting one. The email must:

- Be from a spokesperson of an experiment in the Run Group. Any spokesperson will do.
- Demonstrate some reasonable expectation that the space will be used in the immediate future for an important purpose. No "reserving space for later".
- Identify a person that will be the DST cache manager. The manager will adjudicate conflicting demands for use of the space within the Run Group. This person could be, but need not be, the spokesperson submitting the request.

2.23 What DST cache areas currently exist? Who are the managers?

Run Group	Cache Group Name	Manager	Manager's email	Requester
DVCS	`clasdvcs`	Rustam Niyazov	`rustam@jlab.org`	Stepan Stepanyan
E1	`clase1`	Cole Smith	`lcsmith@jlab.org`	Cole Smith
E1-6	`clase1-6`	Maurizio Ungaro	`ungaro@jlab.org`	Elton Smith
E2	`clase2`	Stepan Stepanyan	`stepanya@jlab.org`	Stepan Stepanyan
E5	`clase5`	Will Brooks	`brooksw@jlab.org`	Will Brooks
E6	`clase6`	Cornel Butuceanu	`ccbutu@jlab.org`	Keith Griffieon
EG1	`claseg1`	Tony Forest	`tforest@jlab.org`	Sebastian Kuhn
EG2	`claseg2`	Maurik Holtrop	`holtrop@jlab.org`	Maurik Holtrop
G2	`clasg2`	Stepan Stepanyan	`stepanya@jlab.org`	Stepan Stepanyan
G3	`clasg3`	Steffen Strauch	`strauch@jlab.org`	Barry Berman
G6	`clasg6`	Dennis Weygand	`weygand@jlab.org`	Dennis Weygand
G7	`clasg7`	Clarisse Tur	`clarisse@jlab.org`	Mikhail Kossov
PrimEx	`primex`	Mark Ito	`marki@jlab.org`	Ashot Gasparian

2.24 Now we have a DST cache partition. How do I fill it up? Empty it out?

When the partition is assigned, a DST cache group name[1] will be associated with it, for example `clasg7`. To write to the DST cache use the `-g` switch of the `jcache` command, for example:

```
jcache -g clasg7 /mss/clas/g7e/dst/dst007.dat
```

To delete a file from the DST cache use the `-g` and `-d` switches, for example:

```
jcache -g clasg7 -d /mss/clas/g7e/dst/dst007.dat
```

The syntax is similar to that for the general cache (see FAQ-2.17). Unlike the general cache, for the DST cache the `-d` switch actually causes the file to be deleted.

2.25 How do I tell how much space we have used on our DST cache?

See the Computer Resources page at

`clasweb.jlab.org/offline/resource_status.html`

Click on "General and DST Cache Usage".

2.26 I am trying to mark a cache file for deletion and get an error about the file being "not in cache group x", even though I know it is in cache group x. What is wrong?

Here is an example of the error:

```
> jcache -d /cache/mss/clas/g2a/data/clas_019763.A01
/cache/mss/clas/g2a/data/clas_019763.A01        Failed: file
/cache/mss/clas/g2a/data/clas_019763.A01 not in cache group clas
```

You need to use the original file name which starts with `/mss` not `/cache/mss`. In the example, the file should have been specified as `/mss/clas/g2a/data/clas_019763.A01`.

2.27 How do I get a list of work and cache disks? Their status?

There is a script that lists all of the CLAS work disk areas using the `df` command. It is `/group/clas/tools/misc/df_clas.sh`.

There is a webpage that lists all of the cache disk areas, general and DST, that are available to CLAS. It is linked from the Computer Resources page at

`clasweb.jlab.org/offline/resource_status.html`

Click on "General and DST Cache Usage".

[1] DST cache group names are not to be confused with Unix groups. The DST cache group is a locally invented software construct. See FAQ-2.19.

2.28 My data consumes too much disk space. What should I do?

Step 1: First determine how much compression you need by considering such parameters as available disk space, statistics needed for an analysis of a subset of your data, ...

Step 2: Based on the number arrived at above you have at least 3 options.

If you need to compress the data by a factor of

a.) 2 to 10:

Consider dropping some banks or gzipping.

`http://www.jlab.org/ccc/mail_archives/CLAS/clas_offline/msg02049.html`

b.) 10 to 35:

Consider different output format (ntuples appear to reduce the cooked data file by AT LEAST a factor of 5)

c.) 70 or more:

Consider restructuring the output to minimize headers and the number of bits used (bit packing) for each entry (if the number of significant digits in your measurement is 5 is there a reason to keep 10?)

`http://www.jlab.org/ccc/mail_archives/CLAS/clas_offline/msg02020.html`
`http://planck.phys.Virginia.EDU/~lcs1h/eg1/dstmaker.html`

The good news is that all 3 of the above have been tried and some code exists.
for (b) see packages/nt10maker or packages/clasroot/rootDST (154 bytes/event)
for (c) see cvs -co eg1_dst (32 bytes/particle)

3 Tape Silo

3.1 What are the Tape Silos? How do they work?

We have two tape silos in the computer center with pass-through capability (a tape in one silo can be mounted in a drive on the other silo and *vice versa*). There are three types of tape drives in the silos:

Drive Type	Single Tape Capacity	Read/Write Speed	Count
StorageTek Redwood	50 GB	10 MB/s	8
StorageTek 9840	20 GB	10 MB/s	10
StorageTek 9940	60 GB	10 MB/s	10

There are nearly 6000 tape slots in each silo. The shortfall from 6000 is due to space required for tape drives. All three types of tapes will fit into any given slot. There is two-armed robot in each silo that retrieves and mounts tapes. It works on one tape motion at a time; the two arms are to minimize travel for any required motion. The robot is capable of 350 mounts per hour.

There are ten computers, or data movers, that handle all requests for tape access. They are named mss1.jlab.org,...,mss10.jlab.org. Each has an independent staging disk.

Data read from the silo is read from a tape drive by one of the tape servers and written to its local stage disk. Next the data is shipped over the network to its destination. For writing the reverse occurs: files are shipped over the network to the tape server, written to its local stage disk and then copied from disk to tape. As such there are two queues, one for tape drives and one for network access. Each complete request, read or write, requires each of these two services.

3.2 How do I find out what is in the tape silo queue?

The computer center maintains a suite of web pages that report on the status of the silos. Find it at

`http://cc.jlab.org/scicomp/JASMine/monitor/jasmine.html`

You can also run the Perl script `$CLAS_TOOLS/misc/silo_status.pl`. You need to be using a version of Perl that has the DBI module installed (see FAQ-7.4). The script works by connecting directly to the MySQL server that holds the database used for silo scheduling. The JLab Computer Center also maintains Java programs that give similar output: `/site/bin/jtsstat` and `/site/bin/jtsstat2`. Both of these will give usage message if the `-h` option is specified.

Finally, you can view the output of `$CLAS_TOOLS/misc/silo_status.pl` on the web; it is updated every 20 minutes. It is linked from the CLAS/JLab Computer Resources page at

`http://clasweb.jlab.org/offline/resource_status.html`

Click on "Tape Queue Listing".

3.3 What are the "dot" files, like `.storeinfo`, I see in various places in the `/mss` tree?

These are MSS system files. They do not correspond to real files stored on tape. They mark their respective directories for particular treatment under JASMine.

`.storeattic` Holds deleted files from the parent directory. Files have a version number added to them.

`.storeinfo` Tells JASMine (MSS) what volume set to use for files written to this directory and all its sub-directories.

`.ts` Deprecated. It was used with the pre-JASMine system (OSM) to determine which OSM server files were read/written from/to.

`.template` Deprecated. Was used by OSM to determine what volume set to use for files written to this directory and all its sub-directories.

3.4 How do we set up the /mss directories for production processing?

We have agreed on a standard configuration for the mss directories for production processing. If run groups need a branch of the tree created, consistent with the structure described below, contact the Computer Center. Any variations will require discussion with Mark Ito.

Here is an example:

```
/mss/clas/g1c/production
|
|--pass1
|  |
|  |--v1
|  |  |
|  |  |--data
|  |  |--skims
|  |  |--misc
|  |
|  |--v2
|     |--data
|     |--skims
|     |--misc
|  ...
|
|--pass2
|  |
|  |--v1
|  ...
...
```

`g1c/production` means g1c non-raw data

`pass1` means processing done on the raw data

`pass2` means processing done on the pass1-proccessed data

`v1` means the first version processing

`v2` means the second version of processing

`data` means the processed data

`skims` means skimmed data

`misc` means everything else

Each node of this tree contains a `.storeinfo` file (see FAQ-3.3), so all files under that directory will go on the same volume set in the silo. Note that there can be further subdivisions under each directory (*e. g.*, `misc/ntuple` or `misc/rootDST`) but all such files will be physically mixed on the same set of tapes. These subdirectories can be produced at anytime by anybody. It is only for the definition of the volume sets and placement of the template files that we need help from the computer center.

3.5 I want to retrieve a file from a silo tape. How do I know if the relevant tape is actually in the silo?

There is a script that will give you the status of all relevant tape volumes when given a list of silo stub files. It is `/group/clas/tools/misc/silo_file_status.pl`. Here is the usage message:

```
usage:
silo_file_status.pl mss_stub_file_1 [mss_stub_file_2 ...]

example:
silo_file_status.pl /mss/clas/tmp/e2/paw_nt/file28.dat
```

and here is the output of the example command:

```
volume location state     bytes storage_group mss_path_name
003874        9 Available  5000 b-tmp         /mss/clas/tmp/e2/paw_nt/file28.dat
```

If the state is reported as "Available" then the tape is in the silo. If the state is "Ejected" then the tape is not in the silo.

3.6 In what order do files from a single jput request get arranged on tape?

```
Date: Fri, 19 May 2000 08:38:20 -0400 From: Andy Kowalski
<kowalski@jlab.org> To: marki@jlab.org Subject: files on tape (jput)
```

A jput will break up a large request into multiple requests of 20 files each. Each request is treated separately and independently. Once you get to the point where OSM stores the files on tape, you are not 100% guaranteed that the files from a single request or multiple requests will be on a single tape in order. It would be very inefficient to do so because you would not be allowing writes in parallel to a volume set. If you request writing to a single volume set at a given time you are still not guaranteed that all the files will be on the same tape. OSM may alternate between multiple tapes. It does this by picking the tape that it can use/get to the fastest.

The only way we can 100% guarantee that the files will be in order (per request) on a single tape (or two if the first is full) is if we make OSM only write to one tape from a volume set at a time and only allow one write to a volume set at a time. The CLAS data acquisition, however, needs to be able to write to multiple tapes in a volume set at the same time in order to keep up with the data rates.

3.7 I got an email that my `jcache/jget` request failed. What do I do now?

The Computer Center recommends a three-step approach:

1. Check that the file you are asking for is actually in the silo. See FAQ-3.5.
2. Re-try the request.
3. If the re-try fails submit a CCPR. See FAQ-9.2.

The most common cause of errors is a failed attempt to read a Redwood tape. Usually this is caused by a faulty tape drive. That is why the re-try has a good chance of working; you will probably be using a different drive (hopefully the bad one gets taken offline at some point). In a very small fraction of the Redwood failures, the tape itself is bad or has been damaged.

The OSM software system that we use to manage the silo does not have very good error reporting; it reports only success or failure. When we switch to using Jasmine (May 2001) this situation should start to improve.

4 Batch Farm

4.1 How do I get started using the JLab Batch Farm?

Read "Jefferson Lab's Batch Farm User's Guide," available in HTML form at

http://cc.jlab.org/docs/scicomp/FarmUsersGuide/FarmUsersGuide.htm

or as a MS Word document at

http://cc.jlab.org/docs/scicomp/FarmUsersGuide/FarmUsersGuide.doc

4.2 How do I check on the status of my farm job?

The computer center maintains web pages that give the status of farm jobs and nodes. Go to http://farml1.jlab.org:9090/. You will find links to tables that display farm status information in various ways.

You can also find a formatted listing of all currently running and pending farm jobs at the CLAS/JLab Computer Resources page at
`http://clasweb.jlab.org/offline/resource_status.html`. Click on "Farm Job Listing".

Finally, look in your `.lsbatch` directory. It is located in your home directory on the CUE. Inside you will see several files associated with each farm job that is running. The job number (obtainable as described above) is encoded into the file names. In particular, the file with the `.out` extension contains a log of the standard output from your farm job. This will help you figure out what your job is doing.

4.3 I have a batch job that is in the UNKWN state. What does this mean?

An UNKWN status for a job means that it is on a machine that cannot be reached. In most cases this is due to a system crash and so the job will not complete. The status, however, will not change until the server is rebooted, which in some cases could take a while. Users should simply ignore these listings and go on assuming that the job is dead and will not finish.

4.4 What are the meanings of the various load statistics kept for the farm nodes?

When you look at the detailed status of an individual farm node you see a section that looks like:

```
 CURRENT LOAD USED FOR SCHEDULING:
          r15s   r1m  r15m    ut    pg    io   ls   it   tmp   swp mem
Total      0.3   0.3   0.7  100%  80.2   602    0 7008  948M  113M 75M
```

The meaning of these parameters is:

`r15s` The 15-second exponentially averaged CPU run queue length.

`r1m` The 1-minute exponentially averaged CPU run queue length.

`r15m` The 15-minute exponentially averaged CPU run queue length.

`ut` The CPU utilization exponentially averaged over the last minute, between 0 and 1.

`pg` The memory paging rate exponentially averaged over the last minute, in pages per second.

`io` The disk I/O rate exponentially averaged over the last minute, in KBytes per second (this is only available when the -l option is specified).

`ls` The number of current login users.

`it` The idle time of the host (keyboard not touched on all logged in sessions), in minutes.

`swp` The amount of swap space available, in MBytes.

`mem` The amount of available memory, in MBytes.

`tmp` The amount of free space in /tmp, in Mbytes.

Finally, you could have looked this up for yourself. See FAQ-4.5

4.5 How do I find the man pages on LSF commands and variables?

Log into the CUE and type:

```
setup lsf
```

You can then access the man pages for `lsfintro`, `lsload`, `lshosts`, `lsinfo` and other related topics.

5 Stand-alone Systems at JLab (mainly Linux)

5.1 I just got an IP address for my computer at JLab. How can I get permission to mount `/group/clas` and the various work disks?

Before we allow nfs access to the CLAS disks, you need to make sure that the following items have been attended to:

- The JLab-mandated security measures have all been taken. See http://cc.jlab.org/docs/security/internal/linux-setup.html.
- The `janed` account has been enabled.
- All user id's and group id's on the local machine match those of the CUE. The easiest way to insure this is to run NIS.

In fact, the first two are required just to get an IP address. If all this is done, contact Mark Ito and let him know that that is the case. He will then ask the JLCC to add the node name to the "claspc" netgroup of NIS. That will allow you to nfs mount the disks. Execute the script `/group/clas/tools/misc/df_clas.sh` to see list of the relevant disks.

5.2 I don't have a copy of the RedHat Linux CD's. Where can I get them?

Two options:

1. Ask a collaborator. There are always a few copies floating around. In particular, Mark Ito is in the habit of ordering them from time to time. You might ask him.
2. Get them from the Computer Center's mirror. The Computer Center has a complete collection (all of the releases) of the CD's on a disk partition that you can read on a CUE machine or mount via NFS. Although not, stickly speaking, images of the CD's the directories contain all of the rpms on the CD's and can be used to do a network install. On the CUE, look in the directory `/mirror/redhat/linux`. For NFS, the server is `mirror.jlab.org` and the partition is `/u/mirror`. For those running autofs, the mountpoint is `/u` and the NIS map is `auto.u.ep`.

6 CVS

6.1 What is CVS?

From the manual: CVS is a version control system. Using it, you can record the history of your source files.

6.2 How do I get a copy of the manual?

There is a nice compendium of CVS information at
http://www.loria.fr/~molli/cvs-index.html. The official home of CVS is at
http://www.cvshome.org/.

6.3 What are the basic CVS commands that I need to know?

checkout: Pulls files out of the repository. You can checkout out individual packages (*e. g.*, `cvs checkout dc`) or the entire tree (`cvs checkout packages`). It is recommended that you checkout a tagged version of the code since the latest version (the default) is not guaranteed to be consistent with the rest of the libraries. The "-r" option is used to specify a tag, *e. g.*, `cvs checkout -r release-1-9 dc`.

commit: Put a modified version of a file into the repository, creating a new revision of that file.

status: Gives the status of a file under CVS control. Right now, if we look at the status of this document we see:

```
> cvs status cms.tex
File: cms.tex           Status: Locally Modified
```

Here "Locally Modified" means that the file has changed since its checkout. Other possible states are:

"Up-to-date": Just as checked out.

"Needs Checkout": The file has not been changed since checkout, but since then a newer revision has been committed to the repository.

"Needs Merge": The file has been modified since checkout and since then a newer revision has been committed to the repository. The merging process is less painfull than you might expect.

log: Gives the complete revision history of a file, including tags and the comments that accompanied each commit.

diff: Does a diff between the current version on disk and the version that was checked out. With switches on it can diff any combination of revisions and the current disk version.

release: Delete a module (a set of files), but before doing so check the status of all files in the module and prompt the user for confirmation before actually deleting them. If there are any files that have a status other than "Up-to-date" or if there are files found that are not under cvs control, the release command will note these before the conformation prompt.

update: Bring a file up-to-date. This may involve bringing in a newer revision, merging a new revision with a modified local file (with or without conflicts), or nothing at all if the file is already up-to-date. You can update with a tag as well (again, recommended). One very useful command is "`cvs -n -q update`". This will print a summary of the status of the current directory without changing any of the files. You will be able to tell which files are modified, which ones are new, and which ones have changed in the repository since they were checked out. See the manual for details.

add: Add a file to the repository. An "add" requires a subsequent "comment".

remove: Remove a file from the repository. The file is not actually deleted. Rather a new revision is created with the attribute "dead". Old versions are thus still available. Subsequent checkouts will skip all dead files.

tag: Put a symbolic tag on a file or module.

6.4 Someone checked in a file with a mistake. Is there a way to delete the version at the end of the trunk?

Not exactly, but there are ways to obtain an equivalent result.

Suppose foo.c has three revisions:

```
1.1----1.2----1.3
```

And we think that 1.3 is way wrong and want to have 1.2 on the end of the main trunk again. What we would really like is then:

```
1.1----1.2                  <--- impossible
```

or

```
1.1----1.2----1.3----1.2    <--- impossible
```

but CVS won't let us do either of these.

One work around goes as follows:

```
cvs checkout abc/foo.c              # check out foo.c from module abc
cd abc                              # cd into the directory
rm foo.c                            # remove the local copy
cvs update -p -r 1.2 foo.c > foo.c  # get revision 1.2 of foo.c
cvs commit -m "Same as 1.2" foo.c   # create revision 1.4
```

So we end up with

```
1.1----1.2----1.3----1.4
        ^             ^
        |             |
        +--identical--+
```

where 1.2 and 1.4 are the same. In the `cvs update` step above, `-p` says write the file to standard output, `-r 1.2` selects revision 1.2, and `> foo.c` redirects the standard output to the file `foo.c`. So the local version is exactly the same as revision 1.2, and can be checked in on top of revision 1.3.

Other variations of the `update` step are:

```
cvs update -p -r release-3-4 foo.c > foo.c
```

which gets a tagged version (with tag `release-3-4`) rather than a specific revision and

```
cvs update -p -D 5/5/2000 foo.c > foo.c
```

which gets the trunk version as of May 5, 2000.

6.5 What's with all these types of tags? Symbolic tag? Branch tag? Sticky tag? Sticky date?

First see the manual for the following terms: branch, revision, module.

The existence of a symbolic tag means that someone has assigned a tag name to specific revisions of a specific set of files in a module.

A branch tag means that the revision is on a particular branch of the tree for that module. Branch tags are a subset of symbolic tags.

A sticky tag or sticky date means that for all future cvs operations on the file the -r option or -D option used in the checkout/update is implied. I. e., the default is now to have the -r or -D option active.

When you cvs checkout a module or cvs update a file or files with the -r or -D options, you get specific revisions of the files. The -r option specifies a revision number or symbolic tag name, *e. g.*, "-r 1.8" or "-r release-2-3". The -D specifies a date/time, *e. g.*, "-D 5/22/97". Without these options you default to the latest revision on the main branch. When these options are used, then the tag/date will be sticky.

Here's an example of a cvs update with the -D option, followed by a cvs status -v:

```
claspc2:dc> cvs update -D 5/22/97 dc_xvst_curve.F
P dc_xvst_curve.F

claspc2:dc> cvs status -v dc_xvst_curve.F
===================================================================
File: dc_xvst_curve.F   Status: Up-to-date

   Working revision:    1.6
   Repository revision: 1.6     .../clas_cvs/packages/dc/dc_xvst_curve.F,v
   Sticky Tag:          (none)
```

```
   Sticky Date:         97.05.22.04.00.00
   Sticky Options:      (none)

   Existing Tags:
        jul03                           (revision: 1.7)
        release-1-1                     (revision: 1.7)
        map                             (branch: 1.4.2)
        premap                          (revision: 1.4)
        release-1-0                     (revision: 1.3)
        start                           (revision: 1.1.1.1)
        arne                            (branch: 1.1.1)
```

The file has a sticky date, since we used the -D option. The revision we get is 1.6, and not 1.7, the head of the main branch at this writing. The -v option to cvs status makes it display all symbolic tags for this file for all time, and which revisions they correspond to and whether or not they are branch tags or not. (Note the consistent tagging scheme ;))

If we update with a plain symbolic tag we get (no -v with the status this time):

```
claspc2:dc> cvs update -r release-1-0 dc_xvst_curve.F
P dc_xvst_curve.F
claspc2:dc> cvs status dc_xvst_curve.F
===================================================================
File: dc_xvst_curve.F   Status: Up-to-date

   Working revision:    1.3
   Repository revision: 1.3     .../clas_cvs/packages/dc/dc_xvst_curve.F,v
   Sticky Tag:          release-1-0 (revision: 1.3)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

If we update with one of the branch tags we get:

```
claspc2:dc> cvs update -r map dc_xvst_curve.F
P dc_xvst_curve.F
claspc2:dc> cvs status dc_xvst_curve.F
===================================================================
File: dc_xvst_curve.F   Status: Up-to-date

   Working revision:    1.4.2.1
   Repository revision: 1.4.2.1 .../clas_cvs/packages/dc/dc_xvst_curve.F,v
   Sticky Tag:          map (branch: 1.4.2)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Note that the branch revision number has an odd number of numbers in it. The revision number of the file itself always has an even number of numbers. The update command retrieves the latest revision on the branch, which happens to be 1.4.2.1.

If we specify a specific revision:

```
claspc2:dc> cvs update -r 1.5 dc_xvst_curve.F
P dc_xvst_curve.F
claspc2:dc> cvs status dc_xvst_curve.F
===================================================================
File: dc_xvst_curve.F   Status: Up-to-date

   Working revision:    1.5
   Repository revision: 1.5     .../clas_cvs/packages/dc/dc_xvst_curve.F,v
   Sticky Tag:          1.5
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Finally, we can specify a branch by its revision number explicitly:

```
claspc2:dc> cvs update -r 1.4.2 dc_xvst_curve.F
P dc_xvst_curve.F
claspc2:dc> cvs status dc_xvst_curve.F
===================================================================
File: dc_xvst_curve.F   Status: Up-to-date

   Working revision:    1.4.2.1
   Repository revision: 1.4.2.1 .../clas_cvs/packages/dc/dc_xvst_curve.F,v
   Sticky Tag:          1.4.2
   Sticky Date:         (none)
   Sticky Options:      (none)
```

The main difference between a branch tag and a normal tag or date is that you can continue to develop along the branch. In the last example above, we can now edit revision 1.4.2.1, and check it in, creating revision 1.4.2.2. Subsequent checkouts using this branch will then get revision 1.4.2.2 since it is now the latest version on the branch.

On the other hand, if we try to do the same thing with a version checked out with a normal tag, then we get an error when trying to commit the modified version. In the following example we update with a normal tag, edit the file, then try to commit it:

```
claspc2:dc> cvs update -r release-1-0 dc_xvst_curve.F
P dc_xvst_curve.F
claspc2:dc> emacs -nw dc_xvst_curve.F
claspc2:dc> cvs commit -m "Added a comment." dc_xvst_curve.F
cvs server: sticky tag ‘release-1-0’ for file ‘dc_xvst_curve.F’ is not a
  branch
cvs [server aborted]: correct above errors first!
```

Branches are useful in several contexts. The manual gives the classic example of a small patch to a major software release. The patch is installed along a branch, so the main branch can go along undisturbed. Another situation where a branch is useful is when you know that you will be making extensive changes to a module, and want to check in intermediate states of the changes as you go along, but anticipate that these intermediate states will not work

well for other developers. If you do your development along a branch, other developers will not get your changes, unless they ask for them specifically, and later you can merge your changes back on to the main branch when they are coherent and useful. The merging is semi-automatic, but that is another subject.

6.6 How do I clear a sticky tag?

For a single file named foo.bar, type:

```
cvs update -A foo.bar
```

For an entire directory, type:

```
cvs update -A
```

from inside that directory.

Note that when you do a `cvs update -A`, you are asking for the latest version on the trunk. If that does not correspond to the stickily-tagged version then CVS will try to merge in the changes between the tagged version and the latest version into your modified version. Even if this happens, CVS will make a backup for you. See FAQ-6.7 for an explanation of merging and where to find the backup.

6.7 What happens when cvs merges my files?

A merge is done when you do a `cvs udpate` on a file that you have changed since you checked it out *and* that someone else has changed and checked back in with his changes. In this case your local version does not reflect the changes made by the other person. CVS tries to insert the other person's changes into your version. This is scary since CVS actually edits your file. But it is not as scary as you might think since (a) a backup is made and (b) nothing gets checked-in.

(a) If the file in question is `foo.bar`, and you originally check-out revision 2.3, the backup is called `.#foo.bar.2.3`. This is the version of the file that has only your changes and not those of the other person.

(b) In this example, the file `foo.bar` now contains both your changes and those of the other person. It's status is now either "Locally Modified" or "File had conflicts on merge". This version has not been checked into the repository. You must do that yourself.

Remember that CVS is trying to make one file with two sets of changes. If the two sets are in entirely disjoint sections of the file, then the merged version simply incorporates both sets of changes. This is the "Locally Modified" case. If the two sets are on overlapping lines of the file, you have the "File had conflicts on merge" case. In the conflicted case, both sets of changes are put into the file, next to one another, with obvious text delimiters indicating which set of changes came from which version. You then edit the file by hand to resolve the conflict.

To do all of this, cvs uses the GNU merge utility. See the `merge(1)` man page for more details.

6.8 How do I access the CLAS CVS repository if I can't mount the CLAS group disk?

From: Maurik Holtrop

Here is a quick summary of how to use the "new and improved CVS" method. This is nice, since it will work from offsite too, which is no longer true for any of the pserver methods, and besides, it is much more robust.

- Define `CVS_RSH` and `CVSROOT`:

  ```
  setenv CVS_RSH  ssh
  setenv CVSROOT <username>@cvs.jlab.org:/group/clas/clas_cvs
  ```

  where `<username>` should be replaced with your CUE username.

- Now, if you have your ssh public/private keys setup properly (and of course, you use a "passphrase" (really you ought to)):

  ```
  ssh-agent tcsh
  ssh-add
  <enter your pass phrase here>
  ```

Now cvs will work like a charm, and will not ask you for passwords. If you did not set the passphrase (shame on you) then you can skip the above last bullet (and anybody that can read the files in your .ssh directory can steal your identity.)

For more information see: "man ssh", and

http://cc.jlab.org/announce/newsletter/CCNLMar02.html

6.9 I was using the "pserver" method to access the CVS repository remotely. That does not work anymore. How do I convert my already-checked-out directories to use the "ssh" method?

There is an expert trick for doing this. In each directory that needs to be converted, `cd` into the `CVS` directory and edit the file called `Root`. There is only one line in that file. Replace

```
:pserver:<username>@jlabs1.jlab.org:/group/clas/clas_cvs
```

with

```
<username>@cvs.jlab.org:/group/clas/clas_cvs
```

and follow the instructions in FAQ-6.8.

7 Databases

7.1 What is a relational database?

In crude terms, a relational database is a collection of two-dimensional tables (*i. e.*, a table has rows and columns) where the columns of one table may contain references to rows of another table. The tables are therefore related. It turns out that a lot of thought has gone into developing this concept and fortunes have been made selling software that will allow folks to put their data into these things and take them out again later.

You can find a nice introduction to data modeling and the relational model at `www.utexas.edu/cc/database/datamodeling`.

7.2 Where can I get modern versions of MySQL at JLab?

Currently there are relatively up-to-date versions of the MySQL binaries, libraries and include files in directories under `/group/clas/mysql`. There are directories for Linux and Sun. The versions that you will find on the JLab CUE under `/apps` are slightly out-of-date as of this writing.

7.3 How do I install MySQL on my Linux box?

To get everything you need, go to `http://www.mysql.com/`. Click on "Downloads". Click on "MySQL n.nn – Stable Release (Recommended)".

For rpm-based systems you want

- The server for i386 systems
- Client programs for i386 systems
- Include files and libraries for development for i386 systems
- Client shared libraries for i386 systems

The "shared" package gives you the C API and the "client" packages gives you the `mysql` command line interface.

To get the Perl DBI, from the aforementioned "Downloads" page, click on "DBI—for connecting to MySQL from Perl scripts". You want DBI-n.nn.tar.gz and Msql-Mysql-modules-n.nnnn.tar.gz. These need to be compiled and installed on your machine. The README in Msql-Mysql-modules has the more comprehensive installation instructions.

7.4 I'm getting an error when I run a Perl script that accesses a database. It says "`Can't locate DBI.pm in @INC`". What's wrong?

You are trying to run a Perl script that needs the DBI Perl module, and you are using a version of Perl that does not have DBI installed. All of our Perl scripts that use MySQL do

their database access via DBI. On the CUE one thing that might work, assuming that the script in question is called `script.pl` and it needs MySQL access, is

```
/apps/bin/mysql script.pl
```

since the versions in `/apps` generally have DBI installed, while those in `/usr/bin` or `/usr/local/bin` generally do not. If you are not on the CUE, someone is going to have to install the DBI Perl module. See FAQ 7.3.

8 General UNIX

8.1 How do I learn more about GNU Make?

The main web page for GNU make is http://www.fsf.org/software/make/make.html. If you go there you will find a brief description along with pointers to distributions and documentation.

8.2 How do I force make to continue trying to make stuff even after it encounters an error?

```
make -k
```

8.3 How do I call FORTRAN from C++?

Fortran routines under UNIX are kinda C routines in disguise. So the calling conventions for Fortran are the same as those for C with the added restriction that all calls in Fortran are by reference, *i. e.*, you pass a pointer to the passed variable rather than a copy of the variable. That's why in Fortran you can always modify the value of the calling routine's variable from the called routine. The compiler uses the pointer, you don't get a choice and you don't get to see it. Neither do you care, in most cases anyway.

In C++, there is a qualifier you can put on function prototypes that force C-like calling conventions rather that the default (and more complicated) C++ calling conventions. Since Fortran is really C in this context that qualifier works for Fortran routines. The qualifier looks like

```
extern "C" { function prototypes go here }
```

There is a simple example of this in `$CLAS_PACK/bankdefs`, the routines are `get_bos_index.cc`, the caller, and `make_bos_call.F`, the callee.

8.4 I created some files and the group ownership does not match my default group. What happened?

Some directories (specifically those in the `/group` tree at JLab) are set up so that all files written into them get the group ownership of the directory itself, *e. g.*,

```
> ls -ld /group/clas/builds
drwxr-sr-x  16 claslib  clas  8192 Apr 14 02:00 /group/clas/builds
```

All files written into `/group/clas/builds` will have `clas` group ownership, even if the user doing the writing is not using `clas` as their default group. The `s` in the group privilege field is the sign that this is the case. On the group disks, the group ownership is important since the quotas on the group disk are by group, independent of the user ownership.

8.5 How do I find out which version of `libg2c.a` g77 is using when it links my binary?

If you type

```
g77 --help
```

you get a list of options to gcc. Of interest for answering this question are:

```
-print-search-dirs       Display the directories in the compiler's search path
-print-libgcc-file-name  Display the name of the compiler's companion library
-print-file-name=<lib>   Display the full path to library <lib>
-print-prog-name=<prog>  Display the full path to compiler component <prog>
-print-multi-directory   Display the root directory for versions of libgcc
```

For example:

```
> g77 -print-file-name=libg2c.a
/usr/lib/gcc-lib/i386-redhat-linux/egcs-2.91.66/libg2c.a
```

9 Random JLab Computer Hints

9.1 Is there a web page for information about doing data analysis at the JLab Computer Center?

The computer center is maintaining a new set of web pages that deal specifically with issues related to analyzing our data. The title of the page is "JLAB Computer Center Scientific Computing" and it can be found at `http://cc.jlab.org/scicomp/`.

There is even an FAQ.

9.2 I am having a problem with a JLab computer system. How do I ask the Computer Center for help and/or information?

Submit a Computer Center Problem Report (CCPR). There are two ways to do this:

1. Send email to `helpdesk@jlab.org`.
2. Fill out a web form at

   `http://mis.jlab.org/mis/ccpr/ccpr_user/ccpr_new_user_request.cfm`

   (if you are inside the JLab firewall).

If you use the second method and select the proper category, the relevant people in the Computer Center will receive email immediately. With the first method someone will read your email and route it according to his or her understanding of your request.

The request goes into a database and gets put on someone's "to do" list. It does not get removed from that list until it is marked "resolved". That makes it slightly hard for them to ignore it. You will receive email everytime the request changes state, whether it is tranferred to someone else's "to do" list, deferred, or resolved.

9.3 I submitted a CCPR, but did not save all of the email's on it. How do I check on its status?

Look at the CCPR browsing page at

`http://mis.jlab.org/mis/ccpr/ccpr_user/ccprframe_user.html`

(if you are inside the JLab firewall). You can do a search on "Requestor Email" to get access all of your reports.

9.4 I have submitted out a CCPR. How do I comment on, revise or clarify my report?

Thanks to Dave Rackley:

If you wish to add anything to a previously submitted problem report before it is completed, or even if it has been completed, you just need to send a reply back to any of the CCPR email messages that have been sent to you regarding that individual report.

You can also send an email message as follows:

```
To:ccpr_reply
Subject:CCPR nnnnn
```

Where `nnnnn` represents the assigned CCPR number.

9.5 How do I subscribe to a JLab email list? Unsubscribe?

See the instructions at http://cc.jlab.org/docs/services/email/How_to_Subscribe.html .

9.6 I can't seem to post messages to a JLab email list. I know I am subscribed; I get everyone else's posts. What is going on?

It's likely that your posts are getting "bounced" and never get distributed. This is one common side-effect of a Computer Center policy aimed at reducing spam on our mailing lists. There is a requirement (implemented in early 2001) that only email from people subscribed to a list are allowed to post messages. Basically the problem is that people are sending messages to lists from email addresses that don't match the one they have on the subscription list; the mail is interpreted as coming from a non-subscriber.

If you have not noticed problems, you are probably OK and are in the majority in that regard. Troubles can come when you change the standard email address that you use.

If you are the administrator of a list, you get the bounced messages. The user who sent it does not! Unless the user is monitoring his email for his own post, he may think he distributed critical information, but in fact did not. You, as administrator have to do something about this! Let the user know and/or post it yourself.

If you are having trouble posting messages, you have two options:

1. Unsubscribe your old address and subscribe with the current one. The following four-step procedure is at least sufficient, if not strictly necessary.

   (a) Find out what your email address is on the subscription list.

   If the email list is called `our_email_list` then send a message to majordomo@jlab.org with the following line as the body of the message (not the subject):

   ```
   who our_email_list
   ```

   You will get, via email, a list of all subscribers. See if you can find your trouble-causing email address among them.

   (b) Unsubscribe from the list.

   If the old unwanted address is `not.me@not.myinst.edu` then send a message to majordomo@jlab.org with the following line as the body of the message (not the subject):

   ```
   unsubscribe our_email_list not.me@not.myinst.edu
   ```

   (c) Resubscribe to the list.

   If you would like to post and receive mail as `me@myinst.edu` then send a message to majordomo@jlab.org with the following line as the body of the message (not the subject):

   ```
   subscribe our_email_list me@myinst.edu
   ```

   (d) Get the basic documentation on majordomo and read it (optional but recommended).

   Send a message to majordomo@jlab.org with the following line as the body of the message (not the subject):

```
help
```

2. Ask your email list administrator to register an alternate address for posting to the list with the Computer Center. This will allow you to post from the alternate address as well as from the address listed on your subscription. You will still only receive email at your subscription address. If your email list administrator does not know how to do this, tell her to contact Dave Buckle of the Computer Center. If you do not know who your email list administrator is, then the aforementioned Mr. Buckle can find out for you.

10 RECSIS

10.1 What is RECSIS? Where is the document?

RECSIS is a main program that acts as a shell for analyzing CLAS BOS-based data. Some of its features are:

- Interactive and batch modes.
- Tcl scripting environment. This allows user-defined functions and interactive access to user-defined variables.
- Hooks for user-supplied routines for global initialization, begin run, event record processing, *etc.*
- Switches for turning on and off all of the reconstruction sub-systems, such as drift chamber, time-of-flight, calorimeters, *etc.*
- Link-time switches for enabling input from the online ET ring.

You can find the document linked from the Offline Stand-Alone Software page, `http://clasweb.jlab.org/offline_soft.html`.

10.2 How do I open two output streams in RECSIS?

From Stepan Stepanyan:

Below is an example of opening 2 files and filling them with different sets of banks:
In the tcl →

```
#
outputfile AllOut.evt PROC 2047;
outputfile DSTOut.evt DST 2047;
#
setc outbanknames(1) all;
setc outbanknames(2) "HEADHEVTEVNTDCPBCCPBSCPBECPBLCPBTRPBTGBI";
#
```

In this example RECSIS will open two files: (1) AllOut.evt, "all" banks, everything in IW area will go out to the file without any selection. (2) DSTOut.evt will have the banks named in the variable outbanknames(2).

In addition, there is an array of flags in `user_evnt.F` that one need to set for specific outputs. For example:

```
      do i=1,10
         err(i)=1
      enddo
c
      nami = mamind(IW,'HEVT')
      ind = IW(nami)
      if (ind .ne. 0) then
        npart=iw(ind+5)/100
        if(npart.gt.0)err(2)=0
      endif
c
      DO i=1,10
         IF(outbanknames(i).ne.'junk')THEN
            ns=i+1
            If(outbanknames(i).eq.'all')Then
               call fwbos(iw,ns,CBANK_WRITE(1:LENOCC(CBANK_WRITE)),ierr)
            Else
               if(err(i).eq.0)call fwbos(iw,ns,outbanknames(i),ierr)
            Endif
         ENDIF
      ENDDO
```

These are actual lines from `user_evnt.F`. The second file will be filled only when the trigger particle was identified (`err(2)=0`).

It may look complicated, but is very efficient and convenient to use. If one wants to have old scheme, then the only thing one needs is:

```
setc outbanknames(1) all;
```

10.3 What is the difference between the `set` and `setc` commands in RECSIS Tcl?

Short answer: you only need to use `setc` when you are setting a string, and that string will be used in a Fortran context. The long answer follows.

Strings are stored differently in Fortran and C. In Fortran, they are stored in `CHARACTER` variables that have some declared length in characters. If the length of the string is less than the length of the `CHARACTER` variable, it is customary to write blanks for the extra characters ("pad with blanks"). In C, strings are an array of `char` variables, where the `char` in the array directly after the last `char` in the string is set to `NULL` ("null-terminated string"). This

causes confusion when trying to input a string through the Tcl shell. The treatment of the string has to be different depending on whether it is stored as a Fortran string or a C string.

`set` is the standard Tcl command for giving a variable a value. In Tcl, all variables are strings and the underlying Tcl libraries are all written in C, so for C strings there is no ambiguity. But in RECSIS, we sometimes us the Tcl shell to input strings that are destined for Fortran `CHARACTER` variables. In these cases you must use the `setc` command. `setc` is a locally supported command, *i. e.*, it is RECSIS-specific, and was written for this one purpose. You can look up the `set` command in the Tcl book. You won't find `setc` there. This is a bit messy, but is necessary since we are mixing C and Fortran so liberally.

`$Id: offline_faq.tex,v 1.127 2004/02/18 16:32:16 marki Exp $`