GNUBatch Release 1

Installation and Administration Guide

Table of Contents

Chapter 1

Introduction

GNUBatch is a fully functioned, high performance Job Scheduler and Management System which is available for a wide range of machines running a Unix-style Operating System.

The original version was written by John Collins at Xi Software Ltd and marketed as **Xi-Batch** (this name may still appear in some diagrams). The names, including many of the program and interface names, have been changed to **GNUBatch** or derivatives and the default installation directories have been changed to GNU standards.

The product consists of a “core product” or “basic product” which contains the scheduling software, command-line and character-based interfaces. Additional options provide for:

- An X-Windows GTK+ or Motif Toolkit Interface. This is not really supported under GNU but the programs do build and run.
- An API for use with C and C++ This has been adapted to work under Java.
- An Interface for MS-Windows using Python and PyQt (this can also be used on other platforms supporting PyQt, including GNU/Linux, as an alternative interface).
- A C and C++ API for use with MS-Windows
- Browser Interfaces

The basic manuals cover the “basic product” and the X-Windows interfaces. Additional supplements cover the other interfaces.

The basic manuals are:

- User Guide - a quick introduction and “cookbook” for use of **GNUBatch**
- Reference manual - a complete description of all components of the basic product.
- Administrator Guide - this manual. Information about installation and customisation of the software.

Also available are:

- API reference manual for Unix and MS-Windows API
- MS Windows PyQt Interface Manual
- Browser Interface Manual

1.1 Documentation Standards

These manuals use various character fonts to indicate different types of information as follows:

`File names and quotations within the text`

`Examples and user script`

`Generic data (where you should put a value appropriate to your own environment)`

`Something you should type`

Program names, whether for **GNUBatch** or standard Unix facilities

Warnings and important advice

1.2 Command Line Program Options

Almost all of the programs that make up **GNUBatch** can take (or require) options and arguments supplied on the command line. As much flexibility as possible is allowed in the specification of these options and arguments. The examples in the manual use which ever notation is clearest.

White space may be inserted into flag arguments as in

```
gbch-r -c COUNT=0 -T 10:16
```

or it may be left out as in

```
gbch-r -cCOUNT=0 -T10:16
```

Single character options may be strung together with one minus sign:

```
gbch-r -mwC
```

or separated, as in

```
gbch-r -m -w -C
```

If mutually contradictory arguments are permitted, the rightmost (or rather the most recently specified) applies.

The ability to redefine option letters has been provided, together with the `+keyword` or `--keyword` style of option. Such options should be given completely surrounded by spaces or tabs to separate them from each other and their arguments, for example

```
gbch-r +condition COUNT=0 --time 10:16
```

In addition, all the commands have an option `-?` or `+explain` (or `--explain`) whose function is to list all the other options and exit.

Chapter 2

Installation

GNUBatch is built directly from the sources and installed by "make install" so there is no special installation routine.

Note, however, that the first time it is installed, the network parameters and the system user will need to be installed, for which the make targets `install-user` and `install-net` are provided.

2.1 Installation directories

The following directories are used to hold component parts of **GNUBatch**. Note that there is a method for relocating parts of them after the product has been built, to use different directories. In conformity with GNU standards, the directories are by default based on `/usr/local`, but you may well want to relocate the spool directory, which can be large.

You can relocate the spool directory and other directories either using symbolic links, or by specifying an alternative location in the "master configuration file" `/usr/local/etc/gnubatch.conf`.

Directory	Function
`bin`	User-level programs
`libexec/gnubatch`	Internal programs
`sbin`	Administration programs
`share/gnubatch`	Data and control files
`share/gnubatch/help`	Help files for programs
`var/gnubatch`	Spool directory for pending jobs

2.2 Installing the system user

Most internal files, and IPC facilities such as shared memory segments and semaphores, are owned by a user `gnubatch`. This user name is like a "sub-super-user" for **GNUBatch** facilities. It is not intended that the `gnubatch` user name should be used extensively, however it (hopefully) serves to distinguish

GNUBatch files from other files.

The only files which are not owned by `gnubatch` are the scheduler process btsched and the network server process xbnetserv, together with some auxiliary programs from the GTK clients, which have to be owned by `root`, as they have to be able to transform to other users.

During the installation, it will be necessary to create user `gnubatch`, If it does not exist. We normally recommend a user id of 51 (i.e. in the "system users" below 100), and your “create new user” routine may need a special option to install this. It is not entirely necessary, however, but we do not suggest that all the standard user-level login directory and contents be included, as the user name is not intended for general use.

When setting up the default group for user `gnubatch`, we suggest that a group such as `daemon`, or `sys` is selected. If you are running **GNUSpool** as well, which shares some facilities, you may want to make it the same group as `gnuspool`.

Chapter 3

Setting up networking

GNUBatch can run with jobs and variables shared between 2 or more Unix hosts, possibly different platforms. To do this, networking needs to be enabled.

Networking also needs to be enabled if you intend to use:

- The API, whether for Unix or MS Windows
- The MS Windows clients

To use any or all of these things, **GNUBatch** must be run in network mode.

It does **not** need to be enabled if you have a standalone computer and only run the standard clients, i.e. the shell, curses, GTK+ or Motif interfaces.

When **GNUBatch** is normally installed, all the networking facilities are included. However it may be started, if required, without networking by providing the `n` option to btsched. However gbch-start does not currently support this, so btsched may have to be started "by hand".

If GNUBatch was previously started non-networked and is restarted with networking enabled, all the jobs and variables intended to be exported will have to be reset, as the exported flag will be turned off by being started non-networked.

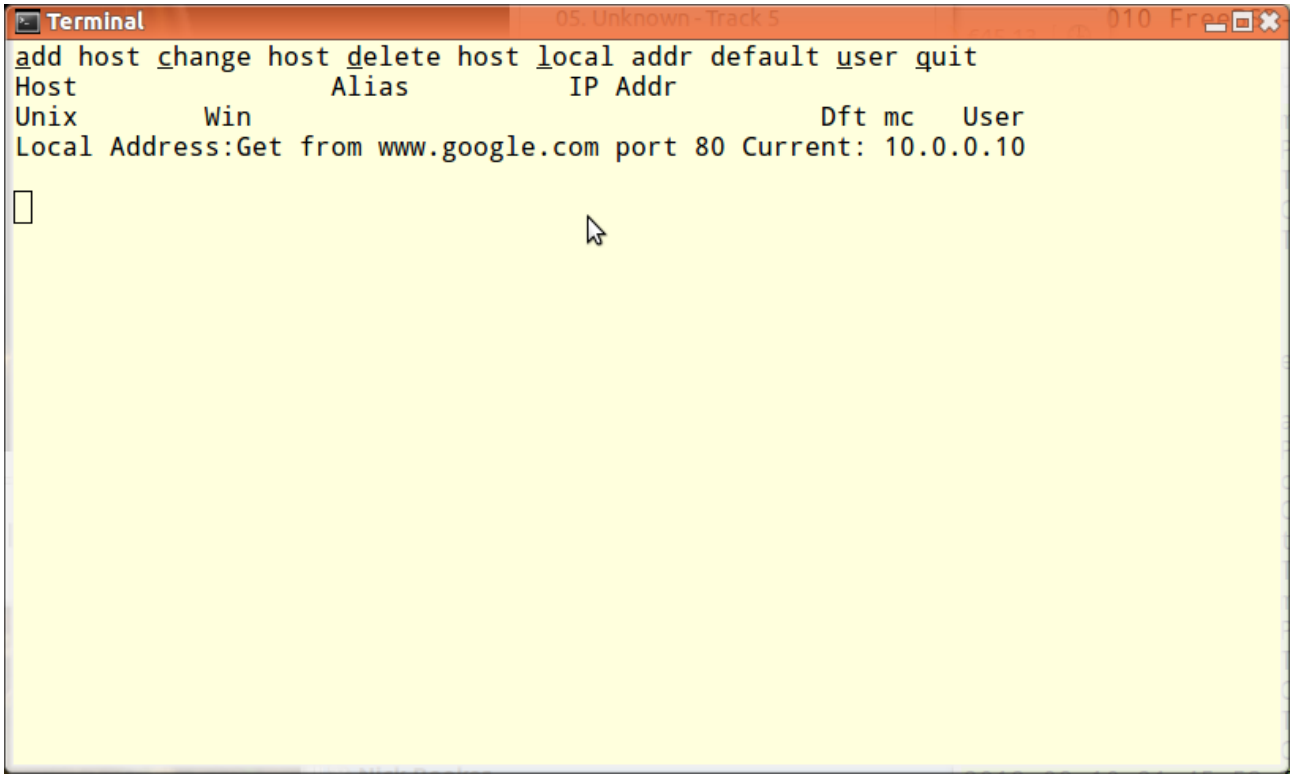
In order for the networking facility to work correctly, three things have to be done:

1. Some standard "service" names need to be inserted in the `/etc/services` file.
2. Hosts may need to be configured in a hosts file `/usr/local/etc/gnubatch.hosts`.
3. The product needs to be started in network mode.

When you install **GNUBatch** normally, these items are all attended to.

3.1 Editing the host file

Unlike earlier versions of **GNUBatch**, you probably only need to edit the host file if:



1. You need to set up a "local address", the IP address by which the host you are installing on is known to the outside world. This is needed as jobs and variables are identified partly by that IP address.
2. You want to give convenient aliases to other hosts you inter-connect with.
3. You want to denote some fixed IP addresses as clients only, not expecting to receive information about jobs and variables (typically for Windows clients).

An additional use, of declaring mappings from Windows user names to UNIX user names, is still supported, but deprecated in favour of the use of the user map file, `/usr/local/etc/gbuser.map` for this purpose.

A special host editor program gbch-hostedit (plus a GUI version of it gbch-xhostedit) is available to edit the host file. This currently still understands and maintains deprecated facilities.

Note that the discussion below is for gbch-hostedit rather than gbch-xhostedit as the latter may not always be available if GTK+ is not available on your system. If you have gbch-xhostedit available, you may want to try using that, hopefully the menu selections will be obvious from the discussion below.

To run this enter:

```
gbch-hostedit -I /usr/local/etc/gnubatch.hosts
```

The `-I` denotes that the file should be edited in place.

With a new installation, the display will take the following form:

The installation script sets up the hosts file to obtain the local IP address by connecting to `www.google.com`.

3.1.1 Setting up the local address

An important aspect of the network code is to obtain the local address.

Jobs and variables owned by the server are distinguished from similar jobs and variables on other servers by the IP address. The server needs to know what its own address is from the point of view of other servers, in order that it can deal with references to its own jobs and variables.

We call this IP address the "local address". It is sometimes not obvious, especially if the server can be referred to by different IP addresses.

Without a hosts file present, **GNUBatch** will try to discover the local address by applying `gethostbyname` to the result of `gethostname`. However this often yields a "loopback" IP address such as `127.0.0.1` rather than something identified to other hosts.

To overcome this, you can set up in the host file:

1. An explicit IP address or host name yielding an IP address to use or
2. A host name or IP address to connect to (usually using the HTTP port) and get the local IP address from `getsockname`.

The latter (more recently introduced) is better, especially on hosts which may have a variable DHCP-assigned IP address which may change between reboots.

To set up a fixed local address, type *l*.

If there is a local address you will be asked:

```
Local address set, do you want to unset it?  [N]
```

Type *y* to unset this prior to setting a new one by typing *l* again.

If there is no local address set, or you have just unset it, you should get the message

```
Do you want to set a local address?  [N]
```

Type *y* and you will get the prompt

```
Local address:
```

Now you can type the host name or IP address to use.

3.1.2 Setting up the local address from a website

Rather than typing in a local address, you can obtain the local address by connecting to a website and noting the local address using `getsockname`.

You can do this once, or you can do it every time (this is best if the IP address may change, due to it being set by DHCP).

Type *w* to select this option, and receive the message:

```
Use Google?  [N]
```

If you reply `N` to this, you will be prompted for another host. If you reply `Y`, then `www.google.com` will be used.

In either event (after checking that the host is valid and has an HTTP server on port 80), you will get the prompt:

```
Always get host that way?  [Y]
```

If you reply `Y` to this, it will set up to always obtain the local address from that host when **GNUBatch** starts, otherwise it will just set the local address IP as if it had been explicitly entered.

3.1.3 Setting up the local address from a connected host

If you have already set up a local host, you can use one of those to set the local address and possibly always do so by moving the cursor to it and typing `L`.

A variety of possible ports and services are tried and the first one to which a connection can be made is used.

Again you will be asked if you always want to get the host that way.

3.1.4 Adding a host

You should add a host:

1. If you want to give a name to an IP address by which the host is referred to.
2. If you want to give a more convenient name to a host rather than a fully-qualified domain name.
3. If you want to arrange for the host to be automatically connected on startup.
4. If you want to say that a given host is only to be used for clients.

We don't recommend that you set up Windows client user names any more, instead use the User Map file `/usr/local/etc/gbuser.map`.

Press "`a`" to add a host, you should get the prompt:

```
Is new entry a Unix host(Y) or Client(N)? [Y]
```

Just press ENTER. Next you should get the prompt.

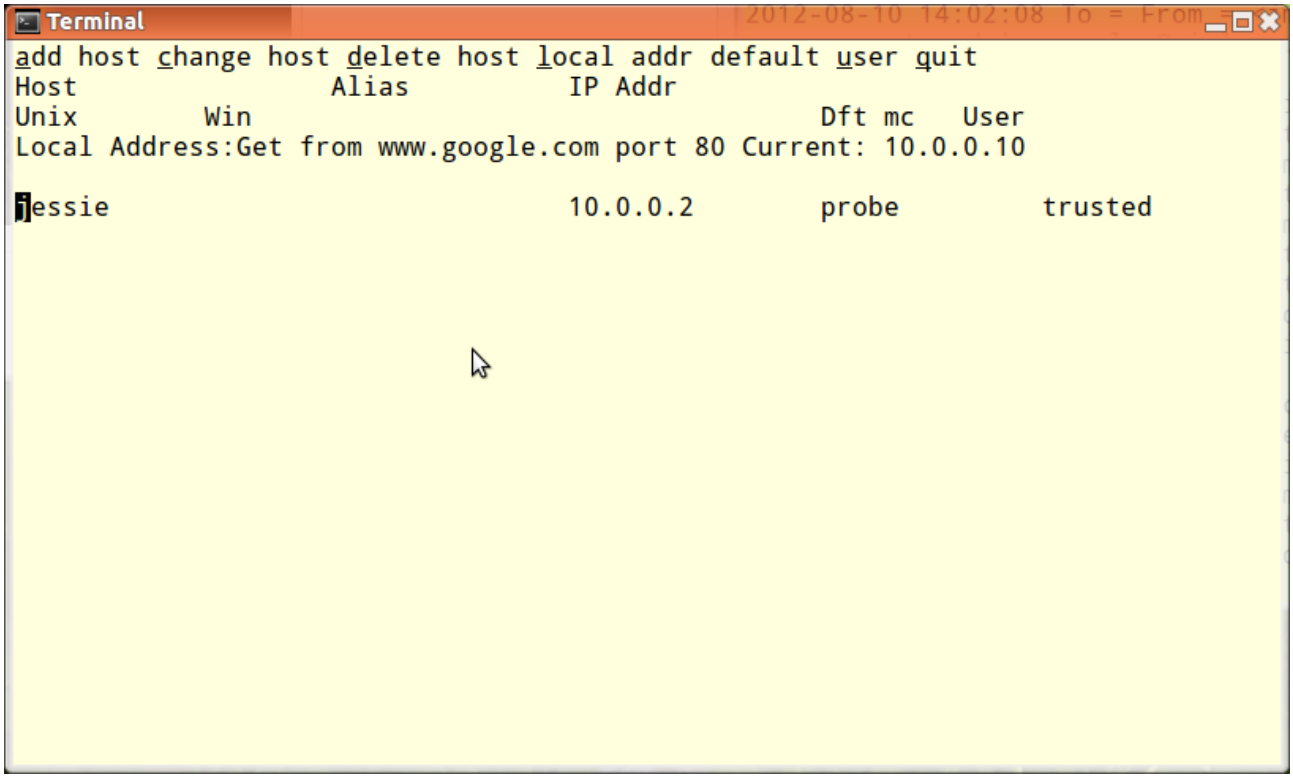
```
Unix host name:
```

Type the name of the host, e.g. on our site, one is called `jessie`. You should then get the prompt:

```
Any alias for jessie:
```

Unless you require an alias, just press ENTER. Next you will get the prompt:

```
Probe (check alive) before connecting?  [Y]
```



Again press ENTER (this checks that the host is on-line before attempting a TCP connection, however in some cases a Firewall etc prevents UDP messages from getting through and this must be avoided.

```
Trust host with user information?  [Y]
```

The function of this has now been deprecated, just accept it.

```
Manual Connections only?  [N]
```

This concerns whether you want the connection to the other Unix host to be attempted as soon as **GNUBatch** is started, or manually by means of gbch-conn. Reply `y` if you require this.

```
Default timeout value OK? [Y]
```

Press ENTER unless you want a variation in the connection timeout. The display should now look like this:

This should be repeated for each host (and should also be repeated on each host so configured). To make any amendments, move the cursor to the relevant host name and type `c` or to delete it type `d`.

3.1.5 Adding a fixed IP address Windows host

Press `a` to add a host name, and this time type `n` to indicate that a Windows host is being added. The prompts are as follows:

```
Specific Windows Host (Y) or `roaming user' (N)? [Y]
```

Press ENTER to denote a specific IP address.

```
Windows client host name:
```

Type the name of the hosts, e.g. “kim”.

```
Any alias for kim:
```

Again, press ENTER unless you want to give an alias.

```
Default user:
```

This is prompting for a default user name to be used for access — the normal UNIX user of that PC, although other users can use it, specifying a password. This can be omitted if required.

```
Password-check user(s)?  [N]
```

This can be turned on to force password-checks on all users if required.

```
Default timeout value OK? [Y]
```

Again, this can be left as the default or not (it is slightly more important, as if the password checks are enabled, it gives the time after which the user will have to log in again with his password if he has not done anything in the meantime).

3.1.6 Adding a DHCP client

Please note that this has been deprecated in favour of using a user mapping file, but is included for compatibility with previous versions. If both are specified, the hosts file, which is read second, will take priority.

This is where the IP may change each time and the recognition is by the user name (or possibly a Windows user name matched to a Unix user name).

Again, press *a* to add a host name and *n* to indicate that a Windows host is being added, and now type *n* again to select DHCP clients.

```
Unix user name:
```

This is the user id under which activities will be run on the Unix host. Be careful to make the case of characters correct (usually all lower case).

```
Windows user name:
```

This is the windows user name, or blank if the same as the Unix user name. It is not case-sensitive.

```
Do you want to specify a default machine name?  [N]
```

This enables the user to specify the (Unix) host name of the usual PC at which the user works. He or she will be able to “log in” at other hosts with the password.

```
Password-check user(s)?  [N]
```

You can set this to insist on a password in all cases.

```
Default timeout value OK? [Y]
```

Again this gives the timeout value. After this time of inactivity, the user will have to log in again.

3.1.7 Quitting and saving

Type `q` to quit and save the newly-created hosts file in `/usr/local/etc/gnubatch.hosts`.

3.1.8 Editing the hosts file after installation

The utility gbch-hostedit (or gbch-xhostedit)may be used to edit the hosts file.

We recommend that **GNUBatch** be halted and restarted after this has been completed.

To invoke it, use the following command:

```
gbch-hostedit -I /usr/local/etc/gnubatch.hosts
```

The `-I` argument denotes that the file is to be edited in place.

3.2 User map file

The user map file provides a mapping between external names, usually Windows user names, and UNIX names.

The file is in `/usr/local/etc/gbuser.map`, and consists (apart from comments introduced by the `#` character) of lines of the format

```
unix-user:windows-user
```

For example:

```
# User mapping file
jmc:john collins
sec:sue collins
guest:default
```

The final entry gives a default user if a named user is not found in the file.

UNIX users not found on the host are silently ignored.

Any number of Windows users may be mapped to the same Unix user.

Chapter 4

Overview of GNUBatch architecture

This is a brief overview of **GNUBatch** for the assistance of administrators. Please see the System Reference Manual for a more detailed discussion.

4.1 Daemon processes

GNUBatch relies on two or four continuously-running "daemon processes" for its operation.

If the product is non-networked (no linked Unix host or hosts, no Windows interface or API), then there are two btsched processes.

- One btsched process handles all incoming requests and is responsible for the overall scheduling.
- The second btsched process actually starts and notes the completion of running jobs.

If the product is networked:

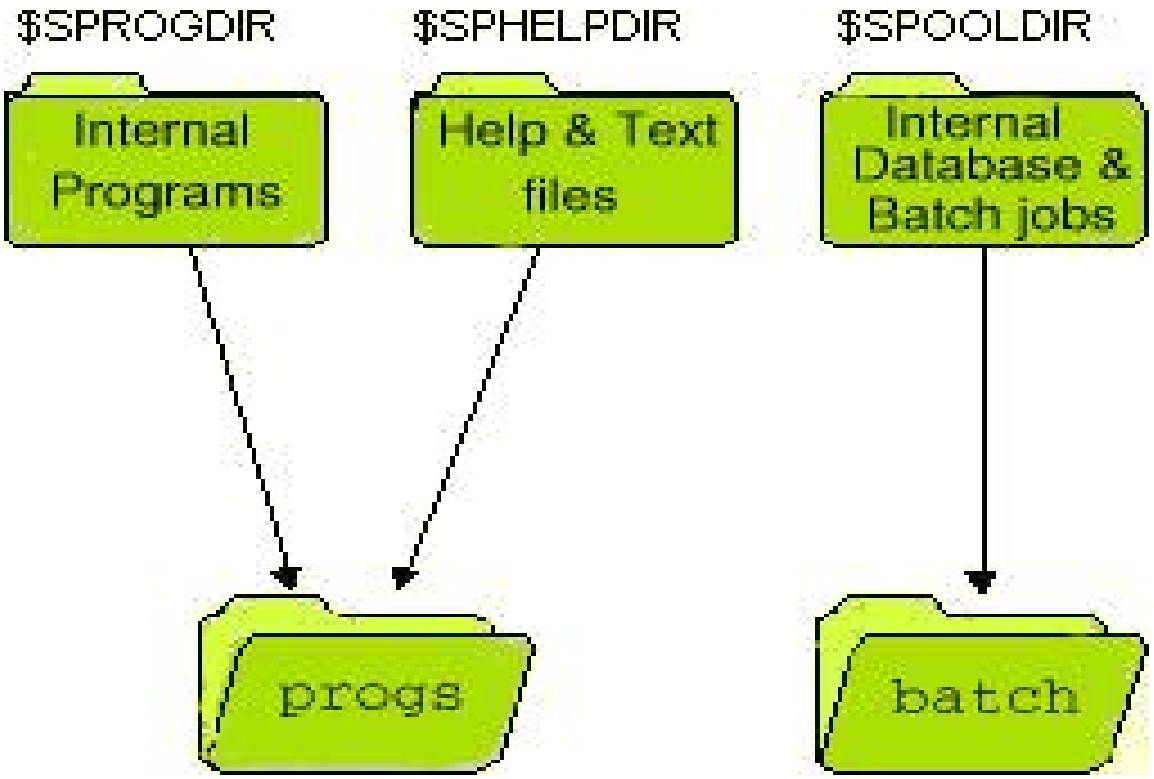
- A further btsched process monitors and processes network traffic between other Unix hosts and the Windows Interface.
- A process xbnetserv handles incoming jobs from the Windows Interface and manages API sessions. (It may "fork off" additional copies of itself for each API session).

These processes are started by running gbch-start and terminated by using gbch-quit. Please do not attempt to start **GNUBatch** in any other way.

Should some system crash ever require you to kill any of the btsched or xbnetserv processes, please do not use "`kill -9`" initially, as this will force immediate termination. Instead use just "`kill`", which will give btsched a chance to release system resources first.

4.2 System directories

GNUBatch uses three logical directories to hold the internal programs and data. These are usually mapped onto two physical directories. A default installation would look like this:



These directories may be relocated by assignment to the three environment variables: `SPROGDIR`, `SPHELPDIR` and `SPOOLDIR`. These environment variable assignments may be placed in the master configuration file, `/usr/local/etc/gnubatch.conf`, to ensure consistency. The default directories are as follows:

Default location	Environment variable	Function
`/usr/local/var/gnubatch`	`SPOOLDIR`	Jobs and other internal data.
`/usr/local/libexec/gnubatch`	`SPROGDIR`	Internal programs
`/usr/local/libexec/gnubatch`	`SPHELPDIR`	Global help and message files

Take care not to assign values to these environment variables arbitrarily; very strange things will happen if one part of **GNUBatch** is using one set of directories and some other part is using another!

4.3 IPC Facilities

GNUBatch uses the "System V IPC facilities" to communicate with itself internally.

- One message queue is used to send requests for the main scheduler process, btsched.
- One shared memory segment saves jobs on the queue. This is periodically written out to the file `btsched_jfile` in the spool directory.
- One shared memory segment saves variables. This is likewise periodically written out to the file `btsched_vfile`.
- One shared memory segment is used to buffer pending requests as the size of messages which may be sent on message queues is limited on many systems.

- One set of semaphores controls access to the shared memory segments.
- One set of semaphores is used for network locking.

The IPC facilities can be displayed by running ipcs. The items in question are owned by `gnubatch` with a key of `0x5869Bxxx`.

The system utility gbch-ripc provided in the installation directory may be used to inspect the IPC facilities especially after a crash and optionally to delete them.

The shared memory used for jobs and variables may have to be reallocated if the number of jobs or variables exceeds the initial amount allocated. On some systems this may be difficult if other software is in use which has exhausted the available shared memory space. If there are difficulties in this area, make sure to start **GNUBatch** with a reservation of the appropriate amount of space. See the description of gbch-start.

On some versions of **GNUBatch** these shared memory segments may be replaced by memory-mapped files. On releases of **GNUBatch** subsequent to November 2004, file locks are usually used instead of semaphores.

4.4 Files used by GNUBatch

The following sections list the files used and created by **GNUBatch**.

4.4.1 Internal spool files

The following **GNUBatch** internal files are held in the spool directory, which by default is `/usr/local/var/gnubatc`

File	Purpose
btufile6	User permissions (the "6" denotes the "major" release).
btcharges6	User charges (the "6" denotes the "major" release) **deprecated**
cifile	Specification of command interpreters
holfile	Days set to be holidays
btsched_jfile	Saved record of jobs
btsched_vfile	Saved record of variables
btsched_reps	Report file holding any messages output by btsched
SPnnnnnnnn	Queued jobs
SOnnnnnnnn	Standard output of pending jobs
ERnnnnnnnn	Standard error of pending jobs
NTnnnnnnnn	Local copy of remote job
btmm_jobs	*If using memory-mapped files rather than shared memory* live memory-mapped file of job queue.
btmm_vars	*If using memory-mapped files rather than shared memory* live memory-mapped file of variables.
btmm_xfer	*If using memory-mapped files rather than shared memory* live memory-mapped buffer file.

The above files are owned by gnubatch. Unused copies of the last four kinds of files may safely be deleted. The nnnnnnnn component of the file name is derived from the batch job number.

Please note in particular the file btsched_reps. This is the system log file and processes such as btsched and xbnetserv send messages to it in the event of error conditions arising. If you have any support questions, please look in this file.

4.4.2 Help and Message files

GNUBatch reads all of its messages from a series of text files (apart from the "help I cannot find the message file" messages). The user may adjust these to tailor the command interface, help and error messages to be suitable for the particular installation. These are *system-wide* message files. It is also possible to set up customised versions for individual users or applications.

The following files are, by default, owned by gnubatch and held in the directory /usr/local/libexec/gnubatch

File	Purpose
btq.help	Screen layout, messages and key assignments for gbch-q
btuser.help	Screen layout, messages and key assignments for gbch-user
btrest.help	Messages and arguments for other user programs
btint-config	Message file for btsched, btwrite and xbnetserv
filemon.help	Messages for file monitor option btfilemon
xmbtq.help	Job and Variable list specifications for gbch-xq and gbch-xmq
xmbtr.help	Job and Variable list specifications for gbch-xr and gbch-xmr
xmbtuser.help	Job and Variable list specifications for gbch-xuser and gbch-xmuser

Refer to the chapters on Configuration and Extensions of both the System Reference Manual and this Administration Guide for details of how to modify these files as may be required.

4.4.3 Configuration files

Various programs allow sets of options to be written into local "configuration files" .gnubatch in the working directory, or a more global .gbch/gnubatch1 file off the user's home directory so that further runs of those programs pick up a different selection of options from the defaults.

These are text files containing sets of options for various programs, which may be edited by the user, but are normally edited by the programs concerned.

4.4.4 Configuration files held in /usr/local/etc

The following files are always held in the system directory /usr/local/etc.

4.4.4.1 GNUBatch Hosts and Clients File

The file /usr/local/etc/gnubatch.hostsis used on networked installations of **GNUBatch** to denote details of the remote hosts and clients to which connection is to be made. This is fully described in the System Reference Manual.

4.4.4.2 GNUBatch Master Configuration File

In order to work properly, the scheduler process and all the other programs must be started with the same environment variables. For convenience, the environment may be initialised for each program by creating a master configuration file /usr/local/etc/gnubatch.conf.

This file contains a list of environment variable assignments. Any environment variables not defined on entry to any of the programs are initialised from this file. Any environment variables used by **GNUBatch** may be included in this file, not just those shown in the example.

For example:

```
SPOOLDIR:/usr1/spool/batch
SPROGDIR:/usr1/spool/bin
MAILER=/usr/lib/sendmail
```

An environment variable declared using the equality sign = will be included in the environment of all batch jobs that are submitted. This may not be wanted for all variables, in particular the directories pointed to by SPOOLDIR, SPROGDIR and SPHELPDIR, as if the batch jobs contain these, they will may be "exported" to other machines on the network for which they would be inappropriate. To avoid jobs inheriting environment variables from the configuration file declare them using the colon.

Please note that the text to the right of the colon or = sign is taken literally; there is no recursive expansion of $name constructs.

4.4.4.3 GNUBatch Static Environment File

If there are a lot of environment variables, the common ones may be saved in a static environment file with only the "differences" from the file held in the jobs.

This file is saved as /usr/local/etc/gnubatch.env as just environment variable assignments.

Alternative files to /usr/local/etc/gnubatch.env can be specified by including the following line in /usr/local/etc/gnubatch.conf:

```
BATCHENV:file1,file2....
```

These files are read in sequence and constitute the new environment.

4.5 Ports and Network Services

GNUBatch also uses 7 entries in the services file, /etc/services. When installed with the default values the entries will look like this:

```
gnubatch        48104/tcp # Connection port
gnubatch        48104/udp # Probe port
gnubatch-feeder 48105/tcp # Feeder port for GNUBatch
gnubatch-netsrv 48106/tcp # External job submission
gnubatch-netsrv 48106/udp # Client access
gnubatch-api    48107/tcp # API
gnubatch-api    48107/udp # API (for wakeup messages)
```

The port numbers can be changed to avoid clashing with those used by other applications, but all hosts in a given installation must use the same numbers if **GNUBatch** connections are to be made.

The installation script will set up entries for the **GNUBatch** TCP & UDP connections in the /etc/services file.

If you are using a name service like NIS instead of the services file, the entries must be copied to that service.

Chapter 5

User administration

GNUBatch maintains a list of users which is generated from the password system (whether using the passwd file or NIS). Hence, each user must normally have a Unix account in order to have a **GNUBatch** account. However a default set of permissions is provided for users who do not have a Unix account.

There are 4 (formerly 5) aspects to the **GNUBatch** user account:

Privileges	Control access to usage and administration functions of the system. For example, the privilege to submit jobs to the queue.
Load levels	Provide a limit on the size of any one job and the total load that that user can place on the system.
Priorities	When there are more jobs ready to run than are allowed these position the “ready” jobs with respect to each other. Facilities exist to specify what priorities each **GNUBatch** user may specify for their batch jobs.
Modes	Specify the default modes that are placed on jobs and variables for the user who creates them. Modes are described in detail in the Variables chapter and the Jobs chapter.
Charges	These are now deprecated.

In addition to these features the user interface can be configured differently for different users and activities.

There are 5 programs for administering user accounts which are:

gbch-charge	A command line utility for querying and adjusting user charges (this is now deprecated)
gbch-uchange	a command line utility for modifying user accounts
gbch-ulist	A command line utility for listing user accounts
gbch-user	An interactive tool for viewing and changing user accounts on character terminals
gbch-xuser	A GTK+ alternative to gbch-user
gbch-xmuser	A Motif GUI alternative to gbch-user

The programs are described in detail in the System Reference Manual.

5.1 Adding New Users

No special action is required to add new users to **GNUBatch** unless the users have to have different permissions from the defaults.

5.2 Removing Users

No special action needs to be taken when users are deleted.

5.3 Setting Privileges

The available privileges are:

Privilege	**Description**
Read admin file	May display contents of administration file showing users, charges and privileges.
Write admin file	May edit the administration file. This makes a user a full **GNUBatch** Administrator since they can grant themselves any other privilege.
Create entry	User may create jobs and variables. This permission is granted by default in the initial configuration when **GNUBatch** is installed.
Special Create	May create jobs (and variables) with different load levels, owners and groups. Can also add, delete and modify command interpreters.
Stop scheduler	May stop **GNUBatch** using the gbch-quit program.
Change default modes	User may change their own default modes. This permission is granted by default in the initial configuration when **GNUBatch** is installed.
Combine user and group permissions	If the user has the same primary group as the job or variable in question, then the permissions are the combination of user and group permissions. This could make a user a super user for managing jobs in their group, without having to grant Write admin file privilege.
Combine user and other permissions	If the user has a different primary group from the job or variable in question, then the permissions are the combination of user and “other” permissions.
Combine group and other permissions	Combine group and “other” permissions. This effectively turns off group handling.

Unless otherwise stated in the table, when **GNUBatch** is installed these privileges are turned off. The currently specified default privileges are applied to all new users.

When **GNUBatch** is installed only the super-user, `root`, and user `gnubatch` are set up as system administrators. These two users are granted all of the privileges (and cannot be deprived of them, attempts to do so will be silently ignored).

Changing users privileges and the default settings can be performed with gbch-user, gbch-uchange, gbch-xuser or gbch-xmuser.

When setting up **GNUBatch** on a machine one of the first Administration tasks should be deciding which extra Unix users to make administrators.

5.4 Setting Default & User Priorities

When several jobs are ready and waiting to run, those with the highest value get started first. There can be constraints which limit the number of jobs that can be running. In this case the higher priority jobs get started, working down the priority until the limit is reached. The remaining jobs have to wait until a running job finished or the limit is raised, when the next highest gets started.

A batch job may have a priority in the range of 1 to 255. Users will usually be restricted to a smaller range between their individual minimum and maximum priorities. When it is installed **GNUBatch** sets the default values to be:

minimum	100
maximum	200
default	150

There is no particular reason for these values. They allow for non-standard users to set up jobs which have higher or lower priority than the normal user population.

When a new policy is decided the default values should be adjusted, then existing user accounts modified as required.

When a job is submitted, it is given the user's default priority unless overridden. It is possible to set a user's minimum, maximum and default priorities to apparently useless values, for example if the default priority is outside the range of the maximum and minimum priorities, the user will always have to specify a priority.

Chapter 6

Other administration matters

6.1 Startup and shutdown of GNUBatch

The user-level programs gbch-start and gbch-quit are provided for the startup and shutdown of **GNUBatch** and gbch-conn and gbch-disconn are provided for attaching and detaching Unix hosts.

Please try to use these utilities wherever possible rather than starting or halting internal processes yourself.

6.2 gbch-start

GNUBatch can be started by just running the program gbch-start. However there are three options to gbch-start which should be worth noting:

The `-j n` and `-v n` options reserve shared memory (or memory-mapped file) space for the given number of jobs and variables respectively. This is often necessary on installations where it is not possible to reallocate shared memory after **GNUBatch** has been started, perhaps because other applications have taken all the available shared memory.

To start **GNUBatch** reserving space for 5,000 jobs and 6,000 variables, use the command:

```
gbch-start -j 5000 -v 6000
```

Another option worth noting here is the `-l n` option, which sets the `LOADLEVEL` system variable to an initial value on startup, commonly zero, which will prevent any jobs from starting until you are ready and all the jobs are in a suitable starting state.

Any user may run gbch-start. If **GNUBatch** is already running, there may be harmless error messages, but otherwise nothing will happen.

6.3 gbch-quit

gbch-quit should be used to halt **GNUBatch**. Any jobs running will be aborted. This may only be run by a suitably-privileged user.

If there are problems with it and some of the processes continue to run and you need to kill btsched and other processes, try first with just "`kill`" and not "`kill -9`" as this will cause the IPC facilities to be deleted.

6.4 gbch-conn and gbch-disconn

gbch-conn is used to connect to Unix hosts set up as for "manual connection" in the hosts file (although more recent versions of **GNUBatch** will accept a simple name of an identifiable host on the network) or disconnected using gbch-disconn.

gbch-disconn disconnects from other Unix hosts previously connected from "either end".

Either are invoked with the host name or alias required. They should return immediately, although the network shutdown may take longer.

Chapter 7

Backup of GNUBatch System

Certain utilities are provided to back up the **GNUBatch** system, by saving the current jobs, variables, user accounts and command interpreter list. You may want to do this when you have set up a complicated schedule of jobs.

All the backup options create a single shell script which, if run, would recreate the relevant items. This may be edited if required.

Jobs, variables, command interpreters and user profiles should be saved in any order, but it is probably best to restore them in the following order to avoid trying to restore items which depend on other items before they are restored.

1. User profiles
2. Command interpreters
3. Variables
4. Jobs

7.1 Jobs

To convert the job list, the utility gbch-cjlist is used. This may be run at any time, even when **GNUBatch** is running, but we would suggest that not too much activity is in progress at the time.

gbch-cjlist is installed in the system binaries directory, by default `/usr/local/sbin`.

You will need to create a directory into which the job scripts are saved.

Here is a suggested routine for saving the jobs:

1. Create a backup directory, for example `/usr/batchsave/May13`.
2. Create a subdirectory within that directory for the scripts, for example you might choose something like `/usr/batchsave/May13/Scripts`.
3. Run gbch-cjlist on the existing job files.

Thus:

```
mkdir -p /usr/batchsave/May13/Scripts
cd /usr/batchsave/May13
gbch-cjlist -D /usr/local/var/gnubatch btsched_jfile Jcmd Scripts
```

The shell command to recreate the jobs will be put in `Jcmd`, and the job script files in `Scripts`.

For more details, and especially the handling of errors, please see the documentation of gbch-cjlist.

gbch-cjlist only considers and saves jobs on the machine on which it is run.

To restore the jobs, just run the shell script `Jcmd` (you may want to edit out some jobs or change parameters on others).

7.2 Variables

Variables are saved in a similar manner to jobs, except no additional directory is required, using the program gbch-cvlist. Only one file is generated, a single shell script of gbch-var commands to recreate the variables.

Assuming that the `/usr/batchsave/May13` directory described above has already been created and selected to save jobs, the following command will save the variables:

```
gbch-cvlist -D /usr/local/var/gnubatch btsched_vfile Vcmd
```

When restoring jobs and variables, you will almost certainly need to restore variables first, otherwise the conditions and assignments in the saved jobs will be rejected.

Please see the documentation of gbch-cvlist for more information, particularly regarding error handling.

7.3 Command Interpreters

Command interpreters are saved in a similar manner to jobs and variables, using the utility program gbch-ciconv.

As with gbch-cvlist, a single shell command file is generated containing commands to recreate the command interpreter file.

To continue the examples above for jobs and variables, the following will create a command file capable of recreating command interpreters:

```
gbch-ciconv -D /usr/local/var/gnubatch cifile Cicmd
```

When restoring the whole system, be sure to restore the command interpreter list and variables before restoring jobs, otherwise the jobs which refer to command interpreters other than the default will be rejected.

Please see the documentation of gbch-ciconv for more information, particularly regarding error handling.

7.4 User permissions

User permissions may be saved in a similar manner to jobs, variables and command interpreters, using the command gbch-btuconv.

As with gbch-cvlist and gbch-ciconv, a shell command is saved which when run will reset the user permissions and defaults for all users.

To continue the examples above for jobs, variables, and command interpreters, the following will create a command file capable of recreating the user permissions.

```
gbch-btuconv -D /usr/spool/batch btufile6 Ucmd
```

When restoring the whole system, it is not necessary, but probably desirable, to restore the user permissions prior to restoring jobs, variables, and command interpreters.

Please see the documentation of gbch-btuconv for more information, particularly regarding error handling.

Chapter 8

System Administration

As opposed to managing users this section is about managing the scheduler. this chapter includes topics like controlling the workload, setting up global entities like command interpreters and keeping audit trails.

8.1 Managing Workload and Auditing

There are seven pre-defined “System” variables known to **GNUBatch**. They are initially set to be owned by `gnubatch` (or `root` if `gnubatch` does not exist) with the default modes which may be reset if desired. These variables may not be deleted or set to an invalid value (e.g. string for numeric variable etc.). They may be included in job conditions or assignments provided that these do not attempt to perform an invalid operation on them.

The variables are:

CLOAD	**GNUBatch** updates CLOAD in real time to show the total load level of all currently-running jobs. This is a read-only variable.
LOADLEVEL	Controls the maximum load of batch jobs that may be running on the system. Jobs can only be started when CLOAD is less than LOADLEVEL. A job can only start if it will not put CLOAD over the limit set by LOADLEVEL. LOADLEVEL may only be set to a numeric value. If the value is increased, then new jobs may start immediately. If the value is reduced, then it is possible that the total load level of running jobs may temporarily exceed it until some of them terminate, however no new jobs will start until the level is no longer exceeded.
LOGJOBS	Specifies where to send output from the Job Audit Trail logging. If the variable holds null (an empty data field) then logging is turned off.
LOGVARS	Specifies where to send log output for the Variable Audit Trail. If the variable holds null (an empty data field) then logging is turned off.
MACHINE	This is a read-only variable containing the current machine name, that can only be referenced as a local variable.
STARTLIM	This variable contains the maximum number of jobs that **GNUBatch** can start at once. The initial value, upon installation of **GNUBatch**, is 5.
STARTWAIT	This variable contains the waiting time in seconds for the next available batch to start if the previous batch set by STARTLIM has not been started for some reason. The initial value upon installation of **GNUBatch** is set to be 30 seconds.

The values of the variables can be read and, except for MACHINE and CLOAD, adjusted using programs gbch-q, gbch-var, gbch-xq and gbch-xmq. These and other variables can also be queried using gbch-vlist.

8.1.1 Managing Total Workload

LOADLEVEL and CLOAD may be used to control the batch workload and avoid conflicts with other activities in a variety of ways. To make the best use of these facilities the load specified for jobs should also be used to reflect their use of resources. Resource hungry jobs should have higher load values then smaller jobs.

8.1.1.1 Running fewer batch jobs in office hours

A batch job can be set up to reduce the value of LOADLEVEL at the start of office hours, to prevent batch jobs slowing down interactive users. Another job can run at the close of office hours to put LOADLEVEL back up to the overnight level.

If jobs have loads in the range 500 to 10000 lowering the value of LOADLEVEL can also be used to prevent some jobs running. For example setting it to 5000 from 20000 will reduce the number of jobs running concurrently but also prevent any jobs over 5000 units from being started.

8.1.1.2 Stopping GNUBatch gracefully

To stop **GNUBatch**, yet allow jobs to complete, perform the following steps:

1. Set LOADLEVEL=0
2. Wait until CLOAD=0
3. Stop **GNUBatch**

This can be done manually, incorporated in a shell script or even set up as a batch job. Batch jobs which stop the scheduler must launch their script asynchronously to avoid killing themselves with the gbch-quit command.

8.1.1.3 Starting activities when Production Work Completes

Set the administration activities up as a batch job to start towards the end of the expected batch work schedule. Specify CLOAD=0 as a pre-condition for the administration job. If there are more than one administration jobs to be run, set them up as a chain of jobs, with only the first one dependent on CLOAD.

The sort of administrative activities that might be performed include back ups, optimising databases, cleaning temporary directories and so on.

This approach is compatible with the idea promoted in the next section. Instead of testing for CLOAD=0 the test might be CLOAD<= 10 for the quoted example.

8.1.1.4 Mixing Administration and Production Jobs

A policy can also be used which will allow small administration jobs to run without being blocked by big production jobs. For example:

- Make the load values for all production jobs go up in increments of 100 units, with the smallest being 100 units.
- Give admin. jobs a load value of 1 unit.
- Setting the value of LOADLEVEL to 20010 will allow a maximum of 20000 units of production jobs. It also lets up to 10 admin. jobs run at any time.
- To stop **GNUBatch** gracefully LOADLEVEL could first be set to 10 to allow admin. jobs to continue running whilst production work finishes. Then when CLOAD drops below 100 set LOADLEVEL to 0 and proceed as above.

8.1.2 Controlling Peak Activity with STARTLIM & STARTWAIT

These variables prevent a large number of jobs swamping a machine or network by starting at the same instant. Any process tends to use a large amount of system resources when starting up. For example: if a user scheduled 400 network I/O intensive jobs to start at Midnight, it is likely that network problems would ensue.

If you observe any resource being swamped then lower the value of `STARTLIM` and/or increase the number of seconds delay specified by `STARTWAIT`. On more powerful machines `STARTLIM` may be increased and/or `STARTWAIT` reduced.

External packages, like alert or performance managers, can query and modify these values using gbch-var. The package must execute the commands as a suitably authorised user.

8.1.3 Keeping an Audit Trail

GNUBatch produces separate output of events suffered by variables and jobs. The output for the Job log is specified by the variable `LOGJOBS` and for variables by `LOGVARS`.

8.1.3.1 Job Logging via LOGJOBS

This variable may only be set to a string value. It should contain a file name, or a program or shell script name starting with a `|`. However, it is vitally important to use `|` with great care so as to ensure the scheduler process cannot be held up by the receiving process.

If a file name is given, it will be taken relative to the spool directory, by default `/usr/local/var/gnubatch`. Thus a file name of `joblog` will be interpreted as `/usr/local/var/gnubatch/joblog`.

The file access modes on the file will correspond to the *read/write* permissions on the variable, and the owner and group will correspond to that of the variable.

If a program or shell script is given, then the `PATH` variable which applied when the scheduler was started will be used to find the program.

Lines written to the file or sent to the program will take the form

```
05/01/99|10:22:43|13741|date|completed|jmc|users|150|1000
```

Each field is separated from the previous one by a `|`, for ease of processing by awk etc. The fields are in the following order(new versions will add fields on the right):

Date	in the form *dd/mm/yy* or *mm/dd/yy* depending on the time zone.
Time	
Job Number	or for external jobs Machine:jobnumber
Job Title	or if not title <unnamed job>
Status code	(listed below) Prefixed by machine: if from remote host
User	
Group	
Priority	
Load Level	

The status codes may be one of the following:

`Abort`	Job aborted
`Cancel`	Job cancelled
`Chgrp`	Group changed
`Chmod`	Mode changed
`Chown`	Owner changed
`Completed`	Job completed satisfactorily
`Create`	Job created (i.e. submitted to queue)
`Delete`	Job deleted
`Error`	Job completed with error exit
`force-run`	Job forced to start, without time advance
`force-start`	Job forced to start
`Jdetails`	Other details of job changed
`Started`	Job started

8.1.3.2 Variable Logging via LOGVARS

This variable may only be set to a string value. It should contain a file name, or a program or shell script name starting with a "`|`".

If a file name is given, it will be taken relative to the spool directory, by default `/usr/local/var/gnubatch`. Thus a file name of `varlog` will be interpreted as the file name `/usr/local/var/gnubatch/varlog`.

The file access modes on the file will correspond to the *read/write* permissions on the variable, and the owner and group will correspond to that of the variable.

If a program or shell script is given, then the `PATH` variable which applied when the scheduler was started will be used to find the program. Lines written to the file or sent to the program will take the form:

```
05/01/99|09:52:43|cnt|assign|Job start|jmc|users|2011|86742|myjob
```

Each field is separated from the previous one by a `|`, for ease of processing by awk etc.

The fields are in the order (new versions will add fields on the right):

Date	in the form *dd/mm/yy* or *mm/dd/yy*, depending on time zone.
Time	
Variable Name	
Status code	(listed below) Prefixed by machine: if from remote host
Event code	see below.
User	
Group	
Value	numeric or string
Job number	if done from job
Job title	if done from job

The status codes indicate what happened, and may be one of the following:

`assign`	Value assigned to variable
`chcomment`	Comment changed
`chgrp`	Group changed
`chown`	Owner changed
`create`	Variable created
`delete`	Variable deleted
`rename`	Variable renamed

The event code indicates the circumstance in which the variable was changed, as follows:

`manual`	Set via user command
`Job start`	Executed at job start
`Job completed`	Executed at job completion
`Job error`	Executed at job error exit
`Job abort`	Executed at job abort
`Job cancel`	Executed at job cancellation

8.2 Setting Up Command Interpreters

The command interpreters are separate entities which are referred to in the job specifications. Every job specifies which command interpreter it will run under. When **GNUBatch** is installed there is normally only one command interpreter set up. It is usually called `sh` and runs the Bourne shell.

Additional command interpreters can be set up to run jobs under the Korn, C, Perl or other shell programs. Any program that reads all of its commands from standard input can be used by a **GNUBatch** Command Interpreter.

Each command interpreter specifies the following set of parameters:

Name A unique identifier which is used, both internally and by user programs, to refer to the command interpreter.

Program Holds the full path and name of the command interpreter program. This can be any program, such as the Bourne shell, `/bin/sh`, or the Korn shell, `/bin/ksh`, that will read commands from standard input.

Arguments Specifies any arguments that are to be passed to the command interpreter when it is invoked. The standard option for Bourne Shell type interpreters is `-s`, which prevents the first "actual" argument from being treated as a file name.

Load Level Sets the default Load level to be given to all jobs running under the command interpreter. Only users with the *special create privilege* may override this default.

Nice Sets the Unix nice value for processing batch jobs under this command interpreter. The default is 24, giving a slightly lower priority from the normal of 20.

Argument 0 When getting a list of processes using a command like ps the batch jobs will normally have the name of the command interpreter program they are running under. Setting the Argument 0 option causes the job title to be used as the process name.

Expand $ constructs Causes the scheduler to expand $-type constructs rather than the command interpreter.

The standard default Bourne shell has the following set up:

Name	`sh`
Program	`/bin/sh`
Arguments	`-s`
Load Level	1000
Nice	24
Argument 0	false (i.e. not set)
Expand $ constructs	false (i.e. not set)

The same program can be used by more than one command interpreter. For example, two command interpreters could be set up using the Bourne shell, but using different options. One could be called sh with the normal arguments. The other could be named `sh_high` with a higher load level and lower nice.

Standard users are not normally allowed to specify different load levels for jobs. They automatically inherit the load from the command interpreter. The mechanism for supporting different types of jobs is to set up a command interpreter for each type. Give each command interpreter a meaning full name.

Here are some examples showing different Programs, Arguments and Load Levels:

Name	reports	updates	admin
Program	`/bin/sh`	`/bin/sh`	`/bin/perl`
Arguments	`-s`	`-s`	`-`
Load Level	1000	3000	1500

Programs gbch-q, gbch-cichange, gbch-xq and gbch-xmq can be used for adding, changing and deleting Command Interpreters. Program gbch-cilist is a command line utility for listing the command interpreters.

To specify the command interpreter when submitting jobs with gbch-r use the `-i` option. Programs PrBtq, gbch-jchange, gbch-xq and gbch-xmq provide facilities for re-specifying a job's command interpreter.

Chapter 9

Configuration and customisation

GNUBatch can be highly configured to restrict or enhance the scope and function of user activities. The System Reference Manual gives the basic information for configuring the user interfaces. This chapter describes some of the applications of this configurabillity and their implementation.

Here is the same example that is used in the System Reference Manual. It shows how program gbch-q might look when configured to take advantage of function keys and show a different set of information in a simplified format.

```
Seq   Job Name             Args       Date/Time         Prog
-------------------------------------------------------------------

  1   start                           08/02/99 10:54     Canc
  2   Process directory    /home      08/02/99 10:54
  3   Process directory    /usr       08/02/99 10:54
  4   Process directory    /tmp       08/02/99 10:54
  5   Collect data                    08/02/99 10:54
  6   Error Handler                   08/02/99 10:54
  7   cleanup                         08/02/99 10:54
  8   setup                           29/01/99 23:01     Done

----F1------F2------F3------F4------F5-----F6-------------------------
   help   enable disable   set    view  view
          run     run      time   job    vars
-------------------------------------------------------------------
```

This configuration could be specific to a particular user or activity. In this case it is taken from a real configuration belonging to user `wally` when using jobs in a queue named `par`. The screen display has been changed as follows:

- The fields `jobno`, `Shell`, `Pri`, `Load`, and `Cond` have been removed. (The queue name is omitted when the view is restricted, it can be explicitly specified).
- The `Time` field is changed from the abbreviated form to show time in full, `Date/Time`. The `Title` field is widened to display more text.

- Argument field added. This shows the differences between jobs which are identical except that they use different data as specified in the arguments.
- Column headings underlined and footer expanded to include function key reminders.

The set of jobs displayed has also been restricted to show just those in the queue named `par`, that belong to user `wally`.

This is what the standard configuration of gbch-q looked like when invoked by user `wally`, on another terminal, at the same time:

```
Seq Jobno   User    Title         Shell   Pri Load Time  Cond      Prog
  0 340     wally   e-mail:dial u sh      150 1000 16:33
  1 734     tony    prog_a        sh      150 1000 06/02           Run
  2 1420    wally   Output Exampl sh      150 1000 29/01           Err
  3 735     tony    prog_b        sh      150 1000 08/02 A_STATUS
  4 736     tony    prog_c        sh      150 1000 08/02 A_STATUS
  5 439     wally   wally         sh      150 1000                 Canc
  6 588     wally   Also Sprach Z sh      150 1000 04/02           Done
  7 564     wally   Daily Update  sh      150 1000                 Run
  8 455     pior    Simple Job    sh      150 1000 11/03           Abrt
  9 309     wally   par:start     sh      150 1000 08/03           Canc
 10 310     wally   par:Process d sh      150 1000 08/03 **Cond**
 11 312     wally   par:Process d sh      150 1000 08/03 **Cond**
 12 313     wally   par:Process d sh      150 1000 08/03 **Cond**
 13 314     wally   par:Collect d sh      150 1000 08/03 **Cond**
 14 315     wally   par:Error han sh      150 1000 08/03 **Cond**
 15 316     wally   par:cleanup   sh      150 1000 08/03 **Cond**
                            -- 9 more jobs below --
================================================================================
```

On the standard configuration jobs owned by users `tony` and `pior` can be seen along with other jobs owned by `wally` which were not relevant to the task in hand.

The following sub-sections describe various ways in which the interface can be configured.

9.1 Default Options

All of the programs in the Base Product can be given or require options on the command line. Default options and values can be specified to save typing or to enforce their use. The defaults can be set up globally, per user, by current working directory or by activity.

9.1.1 Setting Up Defaults

Default options for each command line utility, like gbch-r and gbch-var, is specified using an environment variable of the same name. This holds a string of one or more options and any associated arguments. The interactive programs, gbch-q and gbch-user, each require two environment variables. One holds program options like the command line utilities and the other can point to an alternative help file.

These can be set up as environment variables in the user/process environment. Alternatively an equivalent entries can be set up in the relevant configuration files. The names of the environment variables are generally called keywords. they are always in upper case, whereas the program names are in lower case. Minus signs are converted to underscore.

The program descriptions in the System Reference Manual list the keywords. For example gbch-q is listed as having the keyword `GBCH_Q` for options and `BTQCONF` for the help file.

The chapter on Configurabillity in the System Reference Manual explains the theory. Here are some examples of how defaults can be used for gbch-q:

```
BTQCONF=/usr/local/share/gnubatch/help/gbch-q/prod.help
```

This tells gbch-q to load the file `prod.help` in the specified directory instead of `btq.help` from the `progs` directory. It is a good idea to use meaningful file names, in this case `prod.help` could be the configuration for “Production Jobs”.

If all of these “production jobs” are in specific queues the `-q` option can be used to restrict the view accordingly. For example:

```
GBCH_Q=-Z -q live*,prod*
```

This just selects jobs whose queue name starts with `live` or `prod`. The `-Z` option excludes jobs in the null queue from the job list.

9.1.2 Enforcing Defaults

To enforce a default setting users must not be able to invoke the relevant program directly from the command line. There are two ways of achieving this for most programs:

1. Providing a user interface, like a menu system, that does not expose users to the command line.
2. Concealing the program from the user and providing a wrapper program which the user runs instead. This wrapper program can check the options against the defaults and invoke the intended program accordingly.

9.2 Setting Views of the Job and Variable Lists

There are facilities for selecting which jobs and variables to display. These are completely separate to the modes which dictate who can do what to each job or variable. Whilst these can be used for security purposes they are also just as useful for selecting logical views of the scheduling system.

Users can have logical views imposed upon them or be allowed to select their own. A user can only affect a job or variable if they can see it.

9.2.1 Selecting Jobs by Queue

Programs gbch-q, gbch-jlist, gbch-xq and gbch-xmq have options for queue name selection. The first two use Keywords for setting default values. For example to select only jobs in queue test:

Program	Keyword / Resources
gbch-q	GBCH_Q=-Z -q test
gbch-jlist	GBCH_JLIST=-Z -q test

9.2.2 Selecting Jobs and Variables by Owner and Group

Programs gbch-q, gbch-jlist, gbch-vlist, gbch-xq and gbch-xmq have options for selection by owner and group name. The first three use Keywords for setting default values, the others save a set as requested within the program. For example to select only jobs owned by user `fred`:

Program	Keyword / Resources
gbch-q	GBCH_Q=-u fred
gbch-jlist	GBCH_JLIST=-u fred
gbch-vlist	GBCH_VLIST=-u fred

Similarly for selecting jobs in group `staff`:

Program	Keyword / Resources
gbch-q	GBCH_Q=-g staff
gbch-jlist	GBCH_JLIST=-g staff
gbch-vlist	GBCH_VLIST=-g staff

9.2.3 A real example

The Jobs in the example screen for gbch-q at the beginning of this Chapter were selected by invoking gbch-q with the command:

```
gbch-q -Z -q 'par*' -u wally
```

This could have been set up as the default in a `.gnubatch` file or off the user's home directory `.gbch/gnubatch1` using a line like this:

```
GBCH_Q=-Z -q par* -u wally
```

Alternatively it could have been set up in an environment variable using the commands (assuming the Bourne or Korn shell are in use):

```
GBCH_Q="-Z -q par* -u wally"
export GBCH_Q
```

If these commands are being executed in a general shell script, the current user could be specified by replacing `wally` with `$LOGNAME`. For example:

```
BTQ="-Z -q par* -u $LOGNAME"
export BTQ
```

The `$LOGNAME` is evaluated by the shell not the **GNUBatch** programs. The only place where environment variables like `$LOGNAME` can be used is in the global configuration file `/usr/local/etc/gnubatch.conf`.

9.3 Job and Variable List Formats

The job and variable list formats are described in the System Reference Manual. To quickly create a new layout the format can be re-specified and tested from within gbch-q, gbch-xq or gbch-xmq. New formats can be saved at any time, for use or further development later.

The job list for the example at the beginning of this chapter was produced like this:

1. Create a working directory and change directory to it.
2. Run gbch-q and go into the job list. It will look something like this:

```
Seq Jobno   User    Title          Shell   Pri Load Time  Cond      Prog
  0 340     wally   e-mail:dial u sh      150 1000 16:33
  1 734     tony    prog_a         sh      150 1000 06/02           Run
  2 1420    wally   Output Exampl sh      150 1000 29/01           Err
  3 735     tony    prog_b         sh      150 1000 08/02 A_STATUS
  4 736     tony    prog_c         sh      150 1000 08/02 A_STATUS
  5 439     wally   wally          sh      150 1000                 Canc
  6 588     wally   Also Sprach Z sh      150 1000 04/02           Done
  7 564     wally   Daily Update   sh      150 1000                 Run
  8 455     pior    Simple Job     sh      150 1000 11/03           Abrt
  9 309     wally   par:start      sh      150 1000 08/03           Canc
 10 310     wally   par:Process d sh      150 1000 08/03 **Cond**
 11 312     wally   par:Process d sh      150 1000 08/03 **Cond**
 12 313     wally   par:Process d sh      150 1000 08/03 **Cond**
 13 314     wally   par:Collect d sh      150 1000 08/03 **Cond**
 14 315     wally   par:Error han sh      150 1000 08/03 **Cond**
 15 316     wally   par:cleanup    sh      150 1000 08/03 **Cond**
                              -- 9 more jobs below --
================================================================================
```

3. From the job list enter the "`,`" command. This opens the Job list formats screen, showing the current specification:

```
Job list formats
    Width  Code
        3  n  Sequence
" "
 <      7  N Job number
" "
        7  U  User
" "
       13  H  Title (in full)
" "
       14  I  Command Interpreter
" "
        3  p  Priority
        5  L  Load Level
" "
        5  t  Time or date
```

```
" "
         9  c  Conditions (abbreviated)
" "
 <       4  P  Progress
```

4. The entries can be changed in any order, but it is often easiest to delete unwanted fields first. Move the cursor to the Job number line:

```
Job list formats
    Width  Code
         3  n  Sequence
" "
_<        7 N  Job number
" "
         7  U  User
```

   Press *D* to delete this entry. The result will look like:

```
Job list formats
    Width Code
         3 n  Sequence
" "
" "
         7 U  User
" "
```

5. Repeat the operation for all the lines to be deleted.

6. Now is a good time to add the `Args` column. Move the cursor to the delimiter line above `Time or date`, it is shown here as an underline.

```
" "
        13  H Title (in full)
" "
" "
         5  t Time or date
" "
```

7. Enter the *i* command to insert a new entry above the current line, which is to the left of the delimiter in the job list.

```
 13 H Title (in full)
 " "
 _ 5 t Time or date
 " "
```

   The line is blanked and the cursor moves to the field code column. Enter the code letter for the required data field. In this case type an upper case *A* and press return.

```
        13  H  Title (in full)
" "
```

```
          10  A
           5  t  Time or date
" "
```

Asking for help displays a list of available fields. On a standard terminal there is not enough space to show them all, but they are listed in the section on gbch-q in the System Reference Manual.

A field width is suggested by gbch-q, in this case 10. Type 20 and press return to specify a wider column. gbch-q fills in the remaining field. The result should be:

```
          13  H  Title (in full)
" "
          20  A  Arguments
" "
           5  t  Time or date
" "
```

Edit the `Time or Date` field to be `Date and Time` instead. To do this move to the current entry and add a new field. Specify upper case *T* for the field code and accept the suggested width of 18 by just pressing ENTER. Now delete the original entry.

8. Increase the width of the Title field from 13 to 20 characters. Move to the required line and enter the lower case *w* command. Type in the new width, *20*, and press RETURN.

   The final result should look like this:

```
Job list formats
    Width  Code
        3  n  Sequence
" "
       20  H  Title (in full)
" "
       20  A  Arguments
" "
        5  t  Date and Time
" " <   4  P  Progress
```

9. Type `q` to return to the job list. The program will ask if you want to save the new format. If you say yes it will ask where to save it. If you say no but want to save the changes later you will have to come back into this or one of the other formatting screens.

Back in the main screen the new layout will look like this:

```
Seq   Title               Args       Date/Time          Prog
  1   start                          08/02/99 10:54     Canc
  2   Process directory   /home      08/02/99 10:54
  3   Process directory   /usr       08/02/99 10:54
  4   Process directory   /tmp       08/02/99 10:54
  5   Collect data                   08/02/99 10:54
  6   Error Handler                  08/02/99 10:54
  7   cleanup                        08/02/99 10:54
  8   setup                          29/01/99 23:01     Done
```

```
=============================================================================
```

This is not yet the same as the original example. The configuration was saved in a new help file, which was then edited to give the finished result.

The top line has the word `Title` instead of the string `Job Name`. A simple text editor was used to find and edit instances of the word `Title` in the help file.

The lines at the top and bottom of the screen were also edited using a text editor. By default the specification for the lines at the top of the screen is just this:

```
J1:j
```

and the specification for the footer lines at the bottom is

```
F1:============================================================
```

To add an extra line to "underline" the column headings a second line was added to the `J` specifications like this:

```
J1:j
J2:------------------------------------------------------------
```

To put the Function key reminders at the bottom of the screen replace the two original lines with four that look like this.

```
F1:----F1------F2------F3------F4------F5-----F6------------------
F2:  help enable disable set view view
F3:  run run time job vars
F4:------------------------------------------------------------
```

Setting up the function keys to actually do something is a separate exercise described later in this chapter.

9.4 Help & Error Messages

Almost all of the help and error messages output by **GNUBatch** programs are held in the help files. These messages can be edited with any ordinary text editor.

Each message can be made more informative or shorter. The terminology can be changed to reflect local standards or the whole file can be re-written in a different language. Here is a trivial example of enhancing an error message:

Pressing a key which has no command associated with it in the job list produces this error message:

```
Unknown command - expecting job op
```

The message is defined in the help file, in an error line which looks like this:

```
E200:Unknown command - expecting job op
```

When one of the programs needs a message it looks down the help file for all lines beginning with a particular code. In this case `E200` is the relevant error message.

To enhance the message it could be edited and or have additional lines appended. This version offers some constructive advice:

```
E200:Unknown Key - Expecting a job related command
E200:You probably pressed the wrong key by mistake
E200:Enter a ?  or press F1 to list commands & keys.
```

9.5 Names of Alternatives

Names like those of the different job progress states are referred to as alternatives. There are sets of alternatives for all sorts of parameters like comparison operators, types of I/O redirection and units of time for repeating jobs. Only one alternative can be selected for each parameter.

One of the alternatives is normally specified as the default. When an interactive program, like gbch-q, prompts for selection from a list of alternatives this one will be the default unless a different alternative is selected.

The names of alternatives can be edited. A different default alternative can also be specified. If an alternative like the job progress is used by more than one program then it is important to change the names in all of the relevant help files.

9.5.1 Editing Names

Other batch schedulers use different names for the `Canc` or `Cancelled` state of a batch job. In some cases the name `Canc` has a very different meaning. The name used by **GNUBatch** can be edited to be in line with other packages. For example to change `Canc` to `Held`:

Search the files `btq.help` and `btrest.help` for the strings `Canc` and any variations due to upper case letters. The `Canc` alternative will be specified in lists of alternatives like this one from `btrest.help` for program gbch-jlist:

```
# Progress codes for gbch-jlist
202AD0:
202A1:Done
202A2:Err
202A3:Abrt
202A4:Canc
202A5:Init
202A6:Strt
202A7:Run
202A8:Fin
```

The entry for `Canc` can be edited to

```
202A4:Held
```

The first entry is the default, and has an empty string. This is the alternative for the empty progress state for a job that is waiting to run. Some users do not like a blank field and change it to something like:

```
202AD0:*rdy
```

In this case the completed list would look like:

```
202AD0:*rdy
202A1:Done
202A2:Err
202A3:Abrt
202A4:Held
202A5:Init
202A6:Strt
202A7:Run
202A8:Fin
```

9.5.2 Specifying a Different Default

In program gbch-q, when specifying conditions the not equals, `!=`, comparison is offered as the default. This is specified by the `D` in the alternative code in the help file:

```
# Comparison operations.
# Insert 'D' after the 'A' to denote default.
230A1:=
230AD2:!=
230A3:<
230A4:<=
230A5:>
230A6:>=
```

If the equals comparison is used much more often than not equals the default can be changed by changing the codes for both alternatives. Move the `D` from the code of the `!=` alternative and to the code for the = comparison, like this:

```
230AD1:=
230A2:!=
230A3:<
230A4:<=
230A5:>
230A6:>=
```

9.6 Keys & Commands

The keys and commands of the interactive tools gbch-q and gbch-user can be re-configured. One or more keys is mapped to each command by an entry in the relevant help file which looks like this:

```
K400:?
K401:^
```

```
K402:\e
K403:\s
K404:\r
K405:Q,q,.,\kQUIT
K406:k,\kUP
K407:j,\kDOWN
```

or this:

```
1K501:D
1K503:M
1K504:O
1K505:G
1K507:"
1K510:p
1K511:l
```

The entries which start with a letter `K` are global and those that start with a number apply to a specific sub-screen or option. The component before the `:` specifies the command. The component after the `:` is a comma separated list of key definitions.

9.6.1 Specifying Different Keys

An entry for a command may have one or more keys defined. The default key for requesting help is a question mark. In the `btq.help` file it is specified like this.

```
K400:?
```

If the key definition for the function key F1 is `\kF1` then the F1 key could be set up for getting help instead of the question mark. Change the entry to look like this:

```
K400:\kF1
```

It is likely that not all of your terminals will support function keys properly. In this case both keys can be specified, like this:

```
K400:?,\kF1
```

9.6.2 Disabling Commands

If there is no key defined for a command that command cannot be invoked. Deleting, or preferably converting to comment, the entry in a help file for a key effectively disables that command.

For example the delete job command for gbch-q is specified by the help file entry:

```
1K501:D
```

To disable this command comment the entry out by prefixing it with a `#` like this:

```
# 1K501:D
```

Having disabled a command it is important to edit the help messages to reflect the change. The help associated with delete command for the job list will be adjacent to the key definitions in the help file.

In the case of the delete command the only line that needs changing is from top level help that displays the available commands. This is the line

```
H1:D Delete job
```

If this line contained information about other commands it would need editing. Since it only relates to the delete command it can be commented out:

```
# H1:D Delete job
```

9.6.3 Customising Commands

The function of a particular command can be changed by substituting a macro. Macros are described in the following chapter. First the existing command must be disabled as described above. Then a macro is set up which is invoked by the same key or keys as the original command.

For example the `Delete` command could be replaced by a macro which silently unqueues the job to an archive directory. This could be useful for accident prone users.

Chapter 10

Extending the toolkit

There are various mechanisms for enhancing or extending the functionality of **GNUBatch**, which go beyond customisation of the user interface. Some of these mechanisms are separate products with their own documentation. These are the C programmer's API for Unix and the API for Windows PCs.

The System Reference Manual describes the facilities which are built into the basic product and the Motif GUI option as standard. This chapter gives some applications of these facilities and how to set them up.

10.1 Command Interpreters

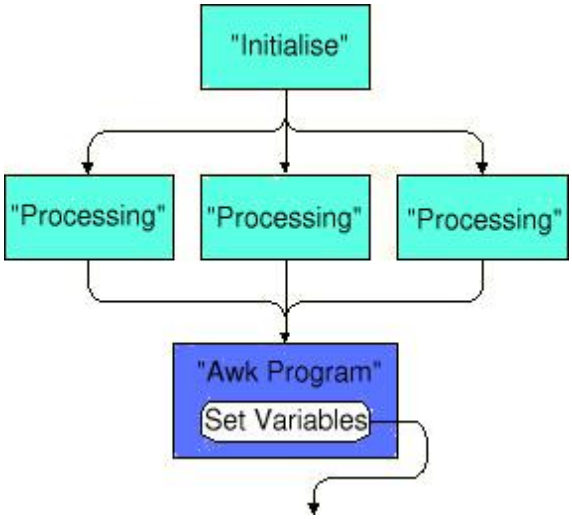
Setting up Command Interpreters is discussed below and in the Reference Manual, but they can also be regarded as a mechanism for enhancing the functionality of **GNUBatch**. This section gives some more ideas for Command Interpreters, and a step by step procedure for setting one up via program gbch-q.

10.1.1 Awk

Any program that can take all the required command from standard input can be set up as a command interpreter. The awk program can do this by specifying the `-f` with an argument of `-` to read standard input. The parameters would look something like this:

Name	`awk`
Program	`/usr/bin/awk`
Arguments	`-f -`
Load Level	1000
Nice	24
Argument 0	false (i.e. not set)

This example has been set up and used on a Sun SPARCstation running Solaris. Other platforms may have different versions of awk and or have different paths to reach them. Some experimentation is likely to be necessary.



These steps were taken to set up the awk command interpreter using program gbch-q:

1. Use the upper case "*X*" command from the job list to open the Command Interpreter screen
2. Move the cursor down to the bottom of the list and give the upper case "*A*" command to add a new command interpreter.
3. Type in the string "*awk*" and press ENTER.
4. Type in the full path and name for the awk program "*/usr/bin/awk*" and then press ENTER.
5. Use the lower case "*a*" command to set up the pre-defined arguments.
6. Type in the string "*-f -*" and press ENTER.
7. The awk command interpreter is now ready to use.

Instead of a shell script the gbch-r command is given an awk program and the name of the file(s) for it to process are specified as job arguments.

By default the output will be e-mailed back to the owner. To send the output to a file use the job I/O redirections.

For example a simple awk program to print the first word from each line containing the string "xi" could look like this:

```
/xi/ {print $1}
```

If the program is in a file called `getxi` and a batch job is to be submitted to run it on `demofile` sending output to `/tmp/xinames` the gbch-r command might be:

```
gbch-r -i awk -a demofile -I ">/tmp/xinames" getxi
```

A real application would be far to big to show here, but here is a suggestion:

"Processing" jobs could produce reports, containing information that impacts upon subsequent jobs.

An awk program could parse the output of the preceding jobs, executing the gbch-var command to set any number of variables, affecting how the rest of the schedule will run.

Alternatively the awk program could be an exception handler, which analyses and corrects problems then restarts the schedule.

10.1.2 SQL

Any Database Product which has a command interpreter that will take commands on standard input can be used in exactly the same way as the awk example. This would be good for people writing SQL, on PCs for example, who do not have an Unix experience.

10.1.3 A dummy Shell

A command interpreter could be written to bypass or dummy_run jobs, by reading the job but not executing any of the commands. To **GNUBatch** it as though the job ran and finished normally. This could be used for testing schedules of jobs without doing any processing.

A shell script for such a command interpreter might look like this:

```
#!/bin/sh
echo Job bypassed
exit 0
```

The script could be expanded to check for job control variable operations that could invalidate testing a job schedule in this manner.

Another version could be set up that always exits with an error code. This command interpreter would be specified to simulate a job running and failing.

10.2 Scripts & Macros

The command line programs for **GNUBatch** enable all job, variable, user and general information to be queried and/or modified. These can be built into new commands using shell scripts or used within user applications and used as macros.

This section gives some more ideas:

10.2.1 Changing several Job Parameters in One Operation

If a number of job parameters are often set to the same value, a macro can be created to apply this specification. For example: A particular type of job may be submitted for testing and when satisfactory changed over to production. The parameters to change for production work are:

Time and Repetition	Run every day after 18:30
Command Interpreter	Change from sh_debug to sh.
Queue name	Same as for testing but prefixed with string P_

A suitable shell script for setting up as a macro would use gbch-jlist to get the current queue name and gbch-jchange to apply the changes. It is as simple as this:

```
QNAME=`gbch-jlist -F "%q" $1`
gbch-jchange -T 18:30 -r Days:1 -q P_${QNAME} -i sh $1
```

gbch-q macro commands pass the currently-selected job number to the macro script.

The gbch-jlist command could apply the prefix to the queue name like this:

```
QNAME=`gbch-jlist -F "P_%q" $1`
gbch-jlist -T 18:30 -r Days:1 -q $QNAME -i sh $1
```

In practice several standard configurations are likely to be required. Rather than use a separate macro for each, the macro could be enhanced to give a list of suitable specifications for the job and ask the user to select one.

10.2.2 Customising an Existing Command

The built in delete command does not do anything about output files produced by jobs. Here is a shell script which deletes the job, then any files produced for standard out and standard error:

```
#!/bin/ksh
# Get the list of I/O redirections for the job.  Separate each
# redirection out - one per line.  Substitute the job id for
# any %d1 in path/file names.  Put results in a temporary file

TEMP=/tmp/jobredirs$$

gbch-jlist -F "%R" $1 | sed -e "s/%d1/$1/g" -e "s/>>/>/g" |\
awk -F, '{ for ( i = NF; i >= 1; i-- ) print $i }' > $TEMP

# cd to the working directory for the job, to cope with
# relative path names for output files.

cd `gbch-jlist -F "%D" $1`

# Attempt to delete the job, and abort if it is running.

gbch-jdel $1

if [ !  -z `gbch-jlist -F "%N" $1` ]
then
    rm $TEMP
   exit 1
fi

# Get the standard output and error file names.

OUTFILE=`awk -F">" ' $1 != 2 && $1 != 0 { print $NF }' $TEMP `
ERRFILE=`awk -F">" ' /^2/ { print $NF }' $TEMP `

# If the files exist then delete them.  Then tidy up.
```

```
if [ !  -z "$OUTFILE" ]
then if [ -f "$OUTFILE" ]
     then rm ${OUTFILE}
    fi
fi

if [ !  -z "$ERRFILE" ]
then if [ -f "$ERRFILE" ]
     then rm ${ERRFILE}
     fi
fi

rm $TEMP
exit 0
```

This example was built and tested on a Sun running Solaris with the version of awk supplied as standard. If your version of awk does not work check the pattern syntax.

To implement the script as a macro:

- Disable the standard delete command by commenting out the key definition:

  ```
  # 1K501:D
  ```

- Specify the path and name of the shell script to be run. If the script is defined as the first macro this might look like:

  ```
  27100P:/usr/macros/deljob
  ```

- Finally set up the macro key definition using the same key as the original delete command. For the first macro this will be:

  ```
  1K701:D
  ```

Chapter 11

Common Problems

This final section deals with common problems which occur from time to time running **GNUBatch**.

11.1 The system log file

At the first sign of any major problem, please check the contents of the system log file `btsched_reps` in the spool directory, by default `/usr/local/var/gnubatch`.

All system processes, such as btsched and xbnetserv append messages to this file before aborting and the messages may make clear what the problem is, so the last few lines of this file may be very informative.

Also check if there are btsched processes running, there should be 3 (with transient extras) if **GNUBatch** is running networked and 2 if it is running networked. There should be 1 xbnetserv process with transient extras to service API requests.

11.2 Files & Directories

One of the most common causes of problems for most products and platforms is changes to files, directory structures, ownerships or permissions. The effects of a missing, duplicate or modified file can be instantly fatal or very subtle. Directories, files, permissions and ownerships should be as described below.

This assumes you are familiar with the output of ls with the `-l` option (use `-ld` if you want to look at the permissions of the directory rather than its contents).

So for example:

```
$ ls -ld /usr/local/bin/gbch-q /usr/local/var/gnubatch
-rwsr-xr-x 1 gnubatch root 215424 Apr 27 17:44 /usr/local/bin/gbch-q
drwx---- 2 gnubatch daemon 4096 May 2 08:11 /usr/local/var/gnubatch
```

The initial block of characters indicates the mode or permissions. The next two fields give the user name by whom the file is owned, `gnubatch` in both the above cases, and the group name by whom the file is group-owned, which is unimportant in nearly all cases.

Normally everything should be world-readable and world-executable in the case of files or in the case of directories, searchable by all users, with the exception of `/usr/local/var/gnubatch`, which is normally restricted to `gnubatch` as above.

All of the user-level binaries such as gbch-q and gbch-r are set-user to `gnubatch`.

Of the internal programs in `/usr/local/libexec/gnubatch`, the following should be set-user to `gnubatch`.

Program	Function
`jobdump`	Copies jobs out to file

The following internal programs should be set-user to `root`.

Program	Function
`bgtkldsv`	Load and restore files for GTK+ interface
`btexec`	Macro execution
`btmdisp`	Message sending
`btpwchk`	Password checking
`btsched`	Scheduler process
`xbnetserv`	Network client manager

Also check that the versions correspond. You might have more than one set of binaries in your `PATH`, so check that the binaries you are actually running correspond with the versions you think you have installed. Running

```
type gbch-q
```

will display what binary is run when you type gbch-q, which may possibly be different from what you think.

You can discover the version number for any binary by invoking it with the `-version` option. Make sure that the version numbers of everything correspond.

You can also discover the version number from the `ident` strings in the file, which you can discvoer from either ident or what if they are available.

11.3 Configuration Files

Errors in the information held by the configuration files can cause fatal or very subtle problems. **GNUBatch** will produce messages about syntax errors.

If a set up problem is suspected check all of the configuration files. For example a network problem may be due to an interaction between the files `/etc/hosts` and the hosts file `/usr/local/etc/gnubatch.hosts`.

11.4 Corruption of Variables

It is possible to purge **GNUBatch** of all the variables by shutting it down and deleting the variable database file `btsched_vfile`. Some users also have procedures where the files in the `batch` directory are moved. These procedures have potentially disastrous side effects and we strongly recommend that you check with us first.

If you restart the scheduler after deleting this file, saved jobs containing conditions and assignments depending on these variables will have the relevant conditions and assignments deleted (a suitable message will be put in the log file in each case). The jobs may therefore start unexpectedly.

11.5 IPC Facilities

GNUBatch uses one message queue to communicate with the scheduler process, btsched. Two shared memory segments are used to hold records of jobs and variables. These records are written out to the files `btsched_jfile` and `btsched_vfile` respectively, after changes occur. A further shared memory segment is used as buffer space for passing job details, as the size of messages which may be sent on message queues is limited on many systems. One semaphore controls access to the shared memory segments, and another is used for network locking.

On some versions of **GNUBatch**, use of shared memory is replaced by memory-mapped files and use of semaphores is replaced by file locking, except for network locking, which still uses a semaphore.

11.5.1 Looking at IPC Facilities

The IPC facilities can be recognised by running ipcs. The items in question are owned by `gnubatch` with a *key* of `0x5869Bxxx`. Using the command `ipcs -o` should produce an output listing resembling this:

```
# ipcs -o

IPC status from <running system> as of Mon Feb 8 12:27:52 1999
T     ID     KEY        MODE       OWNER    GROUP CBYTES  QNUM

Message Queues:
q      0 0x5869b000 -Rrw-------    gnubatch     bin      0     0

T     ID     KEY        MODE       OWNER    GROUP NATTCH
Shared Memory:
m      0 0x50018c83 --rw-r--r--     root    root      1
m      1 0x5869b003 --rw-------    gnubatch     bin      4
m      2 0x5869b002 --rw-------    gnubatch     bin      4
m      3 0x5869b100 --rw-------    gnubatch     bin      4

Semaphores:
s      0 0x5869b001 --ra-ra-ra-    gnubatch     bin
s      1 0x5869b003 --ra-------    gnubatch     bin
```

In this example there are two third party software products using the IPC facilities, as well a shared memory area in use by the operating system. The lines of interest are the ones showing entries owned by user `gnubatch`.

If several products are using IPC facilities there will be more entries displayed than will fit on a screen. The entries appear in the order they were created, as time goes by existing entries will be removed and new ones created. Hence the entries will no longer be in easy to find blocks.

An easy way to see only the entries owned by `gnubatch` would be to pipe the output from the ipcs command through grep. For example:

```
# ipcs -o | grep gnubatch

q      0 0x5869b000 -Rrw-------    gnubatch      bin     0     0
m      1 0x5869b003 --rw-------    gnubatch      bin     4
m      2 0x5869b002 --rw-------    gnubatch      bin     4
m      3 0x5869b100 --rw-------    gnubatch      bin     4
s      0 0x5869b001 --ra-ra-ra-    gnubatch      bin
s      1 0x5869b003 --ra-------    gnubatch      bin
```

The type of each entry is shown in the first column. Message Queues have a letter "`q`" in the first column, shared memory areas have a letter "`m`" and semaphores a letter "`s`".

More recent versions of **GNUBatch** will just show one message queue and one semaphore.

11.5.2 Problem - Cannot Start GNUBatch

When **GNUBatch** is halted normally it removes all of the IPC facilities which it is using. If terminated abnormally the btsched processes will still try to tidy up if possible.

The scheduler cannot be restarted until all old **GNUBatch** processes have been killed and their IPC entries removed. An IPC entry cannot be deleted if there is still a process using or attached to it. This can be a program, like gbch-q or gbch-jlist, not just btsched.

Possible Symptoms:

- Invoking btsched to start the scheduler fails silently.
- Using btstart to start the scheduler fails with an error message about the IPC facilities, I/O or about files other than ones which are held in the batch directory.

Investigation:

- If invoking btsched, run gbch-start instead to get an error message.
- Make sure that the scheduler is stopped.
- Look for btsched, xbnetserv and any user programs with a command like ps.
- Check the IPC facilities with the `ipcs -o` command.

Fix:

- Kill any btsched, xbnetserv and user processes like gbch-q.

- Check the IPC facilitates, for entries owned by `gnubatch`, with the `ipcs -o` command again. Remove any that remain using the ipcrm or ripc commands.
- Run gbch-start to bring the scheduler up, if this fails call for help.

11.5.3 Deleting IPC entries owned by batch

The ipcrm command may be used to delete Message Queues, Shared Memory Segments and Semaphores. It uses the same letters to identify Queues, Memory Segments and Semaphores as the ipcs command. If running `ipcs -o` gives a listing like this:

```
# ipcs -o | grep gnubatch

q      0 0x5869b000 -Rrw-------    gnubatch      bin     0     0
m      1 0x5869b003 --rw-------    gnubatch      bin     4
m      4 0x5869b002 --rw-------    gnubatch      bin     4
m      5 0x5869b100 --rw-------    gnubatch      bin     4
s      0 0x5869b001 --ra-ra-ra-    gnubatch      bin
s      1 0x5869b003 --ra-------    gnubatch      bin
```

then suitable ipcrm commands to remove each entry would be:

```
ipcrm -q 0
ipcrm -m 1
ipcrm -m 4
ipcrm -m 5
ipcrm -s 0
ipcrm -s 1
```

The flag in ipcrm indicates what type of IPC entry to delete, using the same letter as in the first column of the output from `ipcs -o`. A message queue is represented with a letter "`q`" and deleted with a `-q` option and so on.

The number following the flag is an ID number for the queue, memory segment or semaphore. It is unique within the type of IPC, hence there can only be one Message Queue 0, but there can also be one Shared Memory Segment 0 and one Semaphore 0.

Most operating systems will allow more than one entry to be deleted with a single invocation of ipcrm. To delete all 6 entries in one go the command would be:

```
ipcrm -q 0 -m 1 -m 4 -m 5 -s 0 -s 1
```

Please be careful not to delete the wrong IPC entry, possibly belonging to a third-party product!

An easier alternative to ipcrm is the utility program gbch-ripc.

To delete the entries, just type:

```
gbch-ripc-d
```

Ignore any messages, as this program may also be used for debugging purposes, or discard them thus:

```
gbch-ripc-d >/dev/null
```

When it has finished, run

```
ipcs -o
```

again to verify that the IPC facilities have been deleted.

11.6 Messages about key clashes entering gbch-q or gbch-user

If you receive a message like this:

```
State 5 setup error - clash on character ^H with previously-given value
420 and new value 530
```

It means that your help message file contains 2 definitions for the given character, in this case backspace (`^H`). You should look in the help message file (in the case of gbch-q this will be by default `/usr/local/libexec/gnu` and `btuser.help`) for lines of the form

```
K420:  .  .  .
```

and

```
5K530:  .  .  .
```

The “420”, “5” and “530” are all parameters given in the message. You may find that the same character value is defined more than once; remember that (in this example) backspace could conceivably be specified from:

`^H \b` An explicit character definition

`\kERASE` Terminal key settings for erase character etc

`\kLEFT` The *left cursor* key as defined in *terminfo* or *termcap*

The installation procedure attempts to cover cases on the terminal on which you installed **GNUBatch**, but you may run into these difficulties if you use different terminals or key settings.

To overcome the difficulties you may want to rearrange the help message file, or possibly temporarily reassign your terminal keys. Don't forget that you can arrange to automatically select different help message files depending upon the terminal you are using.

11.7 Warning messages about unknown key name

If you receive a message on entry to gbch-q or gbch-user such as

```
Unknown key name 'khome' - ignored
```

then this means that your help message file contains a reference to a key not defined in your *terminfo* or *termcap* file, in this case the `HOME` key. The effect is harmless; however to get rid of the message you should either adjust the file (`btq.help` or `btuser.help`) to remove the reference, in this case to `\kHOME`, or to replace it with the correct character sequence, alternatively you should include the definition in your *terminfo* or *termcap* file.