GNUbatch

John M Collins, Xi Software Ltd
UKUUG Spring Conference
24th and 25th March 2010
History of GNUbatch

- Original version in 1990
- Worked on since then
- GNU version approved in 2009
- Release 2 mostly written but needs completing and debugging.
Agenda for today

- Outline of GNUbatch features
  - Jobs
  - Variables
  - Etc
  - Networking
- GNUbatch 2 features
  - Condition Structures
  - Job Heirarchy
Two approaches

- Layer on existing Unix facilities "cron" and "at".
  - Easy to implement as basic menu package
  - May well achieve requirements

- Ground-up rewrite
Meet a Crontab

Minute: 0
Hour: 9
Day of month: 7-14
Month: *
Day of week: 1-5
User: jmc
Command: run-report
Basic Features 1

- Written from ground up
- Event-driven
- "One-button action" interactive display
- Security features (user permissions etc)
- Variety of interfaces
- Configurable interfaces
- Separate help files (possibly per-user).
Basic Features 2

- Dynamically Updated Queue Viewing
  - Curses Interface
  - X-Windows Interface
  - MS-Windows Interface
  - API

- Transparent networking
  - Jobs and Variables automatically shared
  - Accessed from any point on network
Basic Features 3

- "Shell" interface
  - Many other software items (mail, WP etc) expect this
  - All facilities accessible from command line

- Web Interface
  - Local version (Web Server and GNUbatch same host) or remote version (Web Server and GNUbatch different hosts)
Features 4

- "Jobs" - what to do
  - When
  - How often
  - Files in/out
  - Script

- "Variables" - synchronise jobs
  - "Atomic" test/set
  - Network advertised
Basic Features 5
Job Attributes

- Title
- Queue name (for grouping)
- Next run
- Repeat selection "One-off", every X days/weeks/etc
- Command to run (usually shell, but possibly perl/sql)
- Script to pass (possibly null)
Basic Features 6
Conditions / Assignments

- Specify condition(s) before job can run
- Assignments at start of job
- Assignments at end of job
- End assignments may reverse start assignments
- Different action possible on error/signal
- "Atomic" only one job (network-wide) can "win" race.
Interlocking
Chained example

Note that Jobs and Variables might be on different machines
Conditions

- Variable to be tested
  - Possibly on another host ("critical"?)
  - Read access to owner of job
- Operation (\(= \neq < > \leq \geq\))
- Value (string / number) to be compared
- Combined with AND (in GNUbatch 1)
Assignments

- Variable to be assigned
  - Possibly on another host ("critical"?)
  - Read/Write access to owner of job
- Operation (+ - × ÷ mod or just assign)
- Value (string / number) to be assigned or rhs
- When to apply (start, normal, error, abort, cancel)
- Reverse on exit (or not)
Job Attributes

- Job Number
- Title
- Directory
- Command Interpreter
- Owner / Group
- Access Permissions

- Arguments
- Environment vars
- IO "Redirections"
- Exit code ranges
- Local / Export / Remote Runnable
- Notification flags
- Umask/Ulimit
Job Attributes 2

- Conditions
- Assignments

- Next Time to run
- Repeat specification
- "If not possible"
- Days to avoid
Repeat Styles

Do once and delete

Minutes

Hours

Do once and retain

Days

Weeks

Months (beg)

Months (end)

Years
Months beginning/end?

Holidays from Calendar are 8\textsuperscript{th} kind of day to avoid
If not possible...

- Do when you can (possibly several runs if all missed) usual case
- Catch up by doing just one run
- Skip and just set next time
- Do when can and reset time
Setting job attributes

- **Create job**
  - gbch-r
  - gbch-xr (X-win)
  - Windows interface
  - Web interface
  - API (C or C++, Java possible)

- **Update Job**
  - gbch-jchange
  - gbch-q
  - gbch-xq (X-win)
  - Windows interface
  - Web interface
  - API
Delete / List jobs

- **Delete job**
  - `gbch-jdel`
  - `gbch-q`
  - `gbch-xq`
  - Windows interface
  - Web interface
  - API

- **Also unqueue (esp important)**

- **List / view jobs**
  - `gbch-jlist`
  - `gbch-q`
  - `gbch-xq`
  - Windows interface
  - Web interface
  - API

- **Select / sort as required**
Variable Attributes

- Name
- Value (string / numeric)
- Comment
- Owner
- Group
- Permissions
- Export / "Cluster"
Create/ edit variable attributes

- Create
  - gbch-var
  - gbch-q
  - gbch-xq
  - Web Interface
  - API

- Edit
  - gbch-var
  - gbch-q
  - gbch-xq
  - MS Windows
  - Web Interface
  - API
Delete/list variables

- **Delete**
  - gbch-var
  - gbch-q
  - gbch-xq
  - Web Interface
  - API

- **List**
  - gbch-vlist
  - gbch-q
  - gbch-xq
  - MS Windows
  - Web Interface
  - API

- **Select and group as required**
User Attributes

- Max/Min/Def priority
- Default perms (j/v)
- Max load level (one job)
- Total load level (at once)
- Special create l.l.

- Privileges
  - Read / Write admin
  - Create
  - Special create
  - Vary permissions
  - Stop scheduler
  - OR various perms
Edit User Attributes

- List with
  - gbch-ulist
  - gbch-user
  - gbch-xuser
  - API

- Edit with
  - gbch-uchange
  - gbch-user
  - gbch-xuser
  - API
Load Levels

"Them"

Job 1
Job 2
Job 3
Job 4

BIG Job 1

"Us"

4000

500

500
Each Job has a load level. This is given (usually) by the command interpreter. Privileged users may vary individual jobs.

A system variable (otherwise like the others), called \texttt{LOADLEVEL} gives the maximum load level which may be tolerated. A read-only variable \texttt{CLOAD} gives the current load level.

Jobs ready to run are taken until the total running would exceed \texttt{LOADLEVEL}.
Command Interpreter

- The basic program run by Batch is a command interpreter.
- This designates a program (very often the shell) by a name specified in the job (very often `sh`).
- The script of the job is passed to the command interpreter to run the job (plus any job arguments, I/O, environment etc).
- There may be other command interpreters to designate e.g. Perl, SQL interpreter etc as another program.
Command Interpreter Attributes

- Name of user's choice (should have same meaning throughout network).
- Path name (giving program to be run).
- Default Load level.
- Nice value (system priority).
- Predefined arguments (e.g. -s for shell)
- Flags (set arg0 to title, expand $ constructs)
Command Interpreter Editing

- List with `gbch-cilist`, add/edit/delete with `gbch-cichange`
- `gbch-q`
- `gbch-xq`
- API

Please note assumptions:
- Not changed often
- Same meaning of each name throughout network
Configurable Interface

- Message File provides
  - Help messages
  - Error messages
  - Argument keywords (e.g. `+priority` for `-p` etc)
  - Key command definitions for "curses" programs
  - Prompts
  - Alternatives
  - Numeric Parameters
Argument Handling

- Often tedious to specify the same argument each time. In particular situation might always want same form type, same batch parameters.
- Users might like own preferences e.g. errors in boxes/inverse video etc.
Argument Handling

- First we initialise to defaults, message file / user permissions / standard.
- Next arguments with the same name as the program in upper case, e.g. GBCH_Q for gbch-q from the user's home directory .gnubatch file.
- Next from an environment variable.
- Next from the current directory .gnubatch
- Finally from the command line.
Argument Handling

- Each argument has an "anti-argument" to undo what may have happened before.
- Most shell commands have a `+freeze-current` and `+freeze-home` option to save the current set of options in the home or current directory.
- Interactive commands can do this via a menu option.
- You can actually alter the "search path" and change the order (if so inclined!).
Running on different platforms

- Runs on all current versions of Unix / Linux including Solaris, IBM etc
- Interoperable between different platforms, i.e. Job on Solaris box may be automatically run on AIX box.
- We don't currently operate like that with Windows but we do provide “Windows Agent” to run jobs scheduled by Unix host.
File Monitoring

• Provided as separate "lean mean process" (because it would be expensive to include in otherwise event-driven logic)
• Monitor for file
  - Arriving
  - Deleting
  - Changing (write, grow)
  - Being read
• Do specified operation, e.g. Set variable
Coming soon...

- Release 2 will include:
  - Hierarchy of jobs and associated variables
  - Nested condition structures with many extra features, no limits.
  - “Template” job conditions and assignments.
  - Conditions combined with “AND” / “OR” / “NOT”.
  - Job history which may be used in conditions.
Job Hierarchy

- Can have “super-jobs” with jobs inside them to any level.
- Each “super-job” is a mini scheduler on its own with its own set of variables (but you can refer up and down).
- Conditions stop/start apply to super-jobs and everything down from it.
- Can set whether arguments, environment are inherited etc.
Job Sequence

- Jobs retain relative sequence unless moved.
- Job attributes can be referred to in conditions and assignments.
- Can make relative references e.g. assignment to name of next job whatever that is.
Condition Structures

- No longer part of job but separate object in addition to jobs and variables.
- Contain time, repeats, assignment as individual elements.
- Can put a named set of conditions into a “box” and refer to those.
- Nested and with AND / OR / NOT:
  - Working_hours = 9 to 17 and not 13-14
- Can override bits, e.g use Working_hours but substitute 16:00 on Fridays only.