HOLLAND

ARCHITECTURE

- Broken into two logical components
  - holland-core (single package)
  - holland-plugins (multiple packages)

HOLLAND.CORE

- Scaffold/Framework for building backup interfaces
- No special knowledge of MySQL, LVM, Postgres, etc.
  - (These are all encapsulated in plugins)
- Currently has two frontends: “holland” and “holland_commvault”
  - (Hopefully more interesting interfaces in the future)

HOLLAND.CORE (CONTINUED)

- holland.core consists of 5 main components:
  - config, plugin, backup, command and backports
- Also includes the “holland” script as an interface to various builtin-commands
  - (This will probably be refactored at some point)
- Also includes some utility functions used in plugins and core

HOLLAND.CORE.CONFIG

- Main interface to holland.conf and global-plugin/backupset configurations
- Built on top of Michael Foord's configobj
- Handles merging global and per-backupset configs

HOLLAND.CORE.PLUGIN

- Loads plugins :)
  - (really, loads setuptools “entrypoints”)
- Uses (and currently married-to) pkg_resources
  - as is trac, python-nose, paster, etc.
- http://ianbicking.org/docs/pycon2006/plugins.html

HOLLAND.CORE.PLUGIN (CONTINUED)

- add_plugin_dir(plugin_directory)
- load_first_entrypoint(group, pluginname)
- load_backup_plugin(name)
  - = load_first_entrypoint('holland.backup'.name)
- get_commands(), iter_plugininfo(), iter_dists()

HOLLAND.CORE.BACKUP

- Sets up backup “spool” where backup data is stored
- Loads backupset configs
- Loads plugins
- Dispatches to backup plugins
- Understands “dry-run”
  - Plugins must implement dry-run separately :(

HOLLAND.CORE.BACKUP (CONTINUED)

- backup(backupset_name, dry_run=False)
- Saves a copy of backupset config as ${dst}/backup.conf
- Relies on holland.core.spool.py to manage the backup destination
- raises PluginLoadError or BackupError exceptions

HOLLAND.CORE.COMMAND

- Pluggable command support
- Mostly intended to support subcommands for “holland”
- Also exposes a general run(*args) API usable by external code
- Uses optparse

HOLLAND.CORE.COMMAND (CONTINUED)

```
class MyCommand(Command):

    """${cmd_usage}

    ${cmd_option_list}
    """

    name = 'foo'

    aliases = ['f']

    options = [ option('--foo-the-bar', action=store_true) ]

    description = 'foo a bar'

    def run(self, cmd, options, bar):

    ...
```

HOLLAND.CORE.COMMAND (CONTINUED)

- Command superclass actually calls run via dispatch (*args)
  - introspects arg names from run() signature via inspect
  - (recent argparse patches do this too)
- options[] just set of optparse options
- (should probably be converted to argparse :)

HOLLAND.CORE.BACKPORTS

- Backports of standard python libraries
- mostly for python-2.3 support
- useful for making module behavior more consistent (e.g. subprocess EINTR enhancements from debian)
- holland is currently naughty and loads these in holland.core.__init__ unconditionally

HOLLAND PLUGINS

- Plugins are packaged as python eggs
  - Holland refers to entrypoints as plugins
- Three types of plugins: backup, command and lib
  - (lib isn't really a "plugin")
- Standard location /usr/share/holland/plugins
- Usually part of a holland namespace package

HOLLAND BACKUP PLUGINS

- Three basic methods:
  - __init__(backupset_name, config, directory, dry_run)
  - estimate_backup_size()
  - backup()
- __init__ can be any callable that returns an object

HOLLAND.BACKUP.MYSQLDUMP

- Wraps ‘mysqldump’
- Does size estimations based on information_schema
- supports mysqldump lock modes (none, per-database, full-server, single-transaction)
  - also supports auto-detect for single-transaction
- Database and Table filtering via glob patterns

HOLLAND.COMMANDS

- 90% of commands are in holland-core
- Some proof-of-concept mysql tools in plugins/holland.commands.mysql
- Will probably extend this framework to provide export functionality via sub-sub-commands
  - holland export backupset [backupset-exporter options]

HOLLAND.LIB

- Two main packages here: holland.lib.common, holland.lib.mysql
- Just normal packages that provide functionality for plugins (never used by holland.core directly)
- Some of these could probably be useful outside of holland

HOLLAND.LIB.COMMON

- “Common” functionality
- currently compression and archiving
  - Also a multidict implementation from Ian Bicking
  - Also a portable ‘which’ implementation to find commands
- Fairly simple

HOLLAND.LIB.COMMON - COMPRESSION

- Provides python file-like objects to wrap compression/decompression
- Always expose a usable fileno() attribute
  - Therefore, usable directly by subprocess/fork
- Methods (gzip,bzip2,lzop) hardcoded in dictionary :(
  - (Should be pluggable?)

HOLLAND.LIB.COMMON
(MORE) COMPRESSION

- CompressionOutput = write out compressed files
  - supports inline flag, where it will compress on close ()
- CompressionInput = read and uncompress files
- Based on subprocess, dispatching to a command
- Low python overhead, but portability issues (?)

HOLLAND.LIB.COMMON ARCHIVING

- Only used by holland.backup.mysqlhotcopy
  - (no one uses holland.backup.mysqlhotcopy)
- Intended to abstract zip, tar, or regular directory access
- zip, tar pure python, but perform well-enough
- probably need some cleanup

HOLLAND.LIB.MYSQL

- Used by MySQL plugins
  - Largely holland.backup.mysqldump
- Provides 4 main bits of functionality: client connections, my.cnf parsing, schema filtering and mysql command introspection
- Hard to test independently currently

HOLLAND.LIB.MYSQL.CLIENT
THE CLIENT OBJECT

- Wraps MySQLdb.Connection
  - Can be used as drop-in replacement for MySQLdb.Connection
- Provides useful common functionality used by DBAs on the ‘mysql’ cli tool
  - show_databases, show_tables, flush_*, show_slave_status, show_master_status, etc.

HOLLAND.LIB.MYSQL.FIND SCHEMA FILTERING

- Tightly bound to dbapi :(
- Inspired heavily by Maatkit's MySQLFind class
- Uses inclusion,exclusion lists for databases and tables
- Allows negative and positive searching
  - mysqldump only supports --ignore-table=db.tbl :(

HOLLAND.LIB.MYSQL.OPTION
OPTION FILE PARSING

- http://dev.mysql.com/doc/refman/5.1/en/option-files.html

- Used to merge multiple .my.cnf files
  - Also to write out large mysqldump ignore-table lists for filtering
- my.cnf may use shortest-unique-prefix matching :(
- Other weird behavior of MySQL

HOLLAND.LIB.MYSQL.CLI COMMAND INTROSPECTION

- Derived from DBA scripts
- Mostly used to parse current version of utilities
  - mysqldump.version < mysqlserver.version?
- Also makes it easy to check if a utility (or mysqld) supports some option
  - Although version checks are usually quite sufficient

THINGS TO DO

- Test and document
- Code cleanup
- Fix bugs
- LVM support in high demand (close to being ready)
- Export functionality requested (some good ideas here)

THINGS TO DO TESTING

- holland.lib.common is short and easy to test/ document
  - Code is simple and should be easy to refactor
- holland-core
  - backup, spool needs cleanup and better defined behavior
  - holland.commands could be more robust

THINGS TO DO
END-USER TESTING

- Try crazy stuff with holland backup
  - Stress the table/database filtering for mysqldump
  - Check backup and restore integrity
  - Performance testing - large schemas, compression
- Find broken subcommands or non-intuitive behavior

THINGS TO DO DOCUMENTATION

- User documentation is pretty good from hollandaise 2.0 (Thanks Sweetums :)
- Developer documentation is non-existent (blame Andy)
- How should we document everything?
  - monolithic or manual-per-plugin?

THINGS TO DO
NICE TO HAVES

- Signal support (clean behavior on SIGTERM,SIGHUP,etc.)
- Export functionality - some high level design on gforge
- Refactoring holland.lib.mysql.find to separate filtering from dbapi
- ???