Migrate all the things!

Ryan Weal
Kafei Interactive Inc.
Montréal QC

ryan@kafei.ca
Twitter : http://twitter.com/ryan_weal
Pump.io (open source, ftw!) :
http://comn.ca/ryanweal

In the days before Migrate

- Upgrade the site, simple right?
- What about CCK?
- All the same modules?
- Syntax changes?
- Dropped features?
- New data since the snapshot :(

Let's deal with that new data

- Feeds
  - Rejects duplicates, FUN!
  - Can mutate input, with only 2700 clicks!
- Add XML : now you have two problems!
  - How about we use Regular Expressions to fix the XML?
  - Did anybody write down all the regexes we did?
- Temptation of regexes and the lost steps
- Oops, we screwed up. Let's node_convert.

How do we keep track of all of that?

- Don't bother, use migrate!
- You're going to have to map every field anyway
- Custom code is inevitable
- The mapping class. <3

Main Benefits

- Rollbacks
- Incremental
- Offsets / start points
- Business logic in code
- Multiple-file imports
- Connects to alien databases
- Review interface for client or architect
  - What fields are mapped
  - Progress / error reports

Fringe Benefits, FTW

- Automate like crazy
- Deploy fresh content on a schedule
- Remember that weird thing you did with node references one time? Rebuild it smarter
- Stop architecting around warts
- Create fresh new sites with the latest cool stuff

This is awesome, but boss wants upgrade

- Ughhhhhhh
- The delete_all module from 2bits
- Reset your db counters on your tables
- Offset your migration to start at a certain number

Prefab migrations

- Drupal to Drupal framework « wizard api » (dev version)
- Übercart → Commerce
- Wordpress to content type
  - Use the WP backup file
  - Remap image URLs? :(
- Maybe your Typo3 was a mistake?

Custom migrations

- Defined as a migration class
- Therefore uses OOP
- Ideally you want to extend existing classes
  - D2D templates for D5, D6, D7!
    - Yes you can D7 → D7!
    - D6 passwords? No problem.
  - Übercart → Commerce, etc.
  - Wordpress : you could fix the image paths!
  - Only write the custom field parts!

Registering your classes

- Your module will need to register your new classes
  - Previously automatic, proven that was problematic!
  - Forcing a refresh on cache clear is heavy but good for you
- Class will extend basic templates or prefab ones

Binding to your sources

- Use standard Drupal database config as a secondary db
- XML adapter
  - Rotate through multiple files! :D
- MS-SQL
- Oracle
- Etc...

Binding to your destination

- Use a class that supports your target
- Generic classes are out there
- See drupal contrib for a list of classes applicable to migrate

Minding the map

- Automatically keeps an index of the old database IDs to new IDs
- All the migrations know about each other's new IDs! :D
- Run your migrations in the right order
  - Specify the order in which they should be run
  - Group the migrations, batch the entire group
- XML wants to be indexed as char, might need to re-register
  - Blow away the two tables
- Might need to explicitly prevent joining across different databases...

Fields and queries

- List out all of your fields you will migrate and associate with source field
  - Source field can be optional, you might not know (or care!) where it comes from at this stage
  - For XML you put your Xpaths here
- Many fields are supported by default
  - Field modules *should* include classes to support structure but (ahem, *date*) sometimes they are out-of-date and/or broken
- Unsupported fields can be mapped, we will deal with them later.
- You can set some DNM / do not map (for UI only)

Base Query (1)

- Pull as many fields as you can in the base query, map to fields (save problematic fields for prepare [2] or prepareRow [3])
  - Eventually MySQL will explode
  - Deal with it : run locally!!
- Too many joins and you're going to have a bad time
- If you write the query and get odd errors verify your syntax

PrepareRow Query (2)

- Uses field handlers
- Simple key-value pairs
- Try just throwing data at it and see if it works!
  - $row->thing = thing;
- Look at the row here and reject it if you want (return FALSE)
- No dealing with XML in this stage, it would cause memory problems
- Get stuff and save it for the next stage
  - $row->tmp_junk = tmp_junk;

Prepare Query (3)

- Fully expanded Drupal objects that we know and love
  - Deal with all your unsupported fields here
  - Date field not working? Who cares?! I can deal with an expanded Drupal array!

Complete Query (4)

- Dirty post-processing work
  - calculating commerce order totals from the migrated items in the order
- Affecting unrelated things in the system, live dangerously!
- Really you probably don't need this in most cases

Running the migration

- Use drush, apache/nginx gets in the way
- Manages memory and long running processes automatically
- Get cozy with drush_print_r();
- Interrupted? Use Reset!

Migrate resources

- The migrate example module has Beer
- When you are done with Beer, then try Wine
- Documentation on drupal.org
  https://drupal.org/node/415260
  - Print version grabs sub-pages ;)
- Review the Doxygen summaries :
  http://drupalcontrib.org/api/search/7/migrate
- My blog post! Contains drush commands and example code :
  http://www.verbosity.ca/working-drupals-migrate-module