Drupal Capistrano - CUL deploy.rb

Here is an example deploy.rb with a bunch of features specific to deploying Drupal on CUL servers. (You'll need to change all the TODO's)

```
require 'capistrano/ext/multistage'

set :application, "features.library.cornell.edu"  # TODO - name your application

default_run_options[:shell] = '/bin/bash'

# based on:
# http://guides.beanstalkapp.com/deployments/deploy-with-capistrano.html
# http://www.58bits.com/blog/2013/03/23/deploying-drupal-with-capistrano
# https://github.com/antistatique/capdrupal

# TODO: list of modules to enable
# drush pm-list | grep Enabled | grep -o '(.\+)' | grep -o '[^()]\+' | tr '\n' ' ' && echo ""

set :enabled_modules, File.open("drupal_config/enabled_modules.txt", "r") { |file| file.read.split(" ") }
# override these in production
set :disabled_modules, File.open("drupal_config/disabled_modules.txt", "r") { |file| file.read.split(" ") }
set :default_theme, File.open("drupal_config/default_theme.txt", "r") { |file| file.read.split(" ") }

#SSH and PTY Options
default_run_options[:pty] = true
ssh_options[:forward_agent] = true
set :use_sudo, false
#set :port, 5144

#On Mac OS X
ssh_options[:compression] = "none"

#Application settings
set :user, ENV['USER']  # can also come from ~/.caprc
set :runner, ENV['USER']  # when sudo-ing use this user
set :runner_group, "lib_web_dev_role"
set :admin_runner, ENV['USER']  # when sudo-ing during :setup use this

set :php_user, "apache" # user php runs as

set :stages, ["local", "staging", "production"] # TODO - define different stages
set :default_stage, "staging"

set :keep_releases, 5
set :drush_cmd, "drush"

set :scm, :git
set :repository,  "git@git.library.cornell.edu:features_test_library_cornell_edu.git" # TODO - your git repo
here
set :branch, "master"

# prevent errors from missing javascripts,stylesheets,images directories
set :normalize_asset_timestamps, false

# now use the regular cap deploy instead of drupal
#before 'deploy', 'deploy:drush:site_offline', 'drupal:db:backup', 'git:push_deploy_tag'
before 'deploy' do
  git.push_deploy_tag
  drupal.drush.site_offline
  drupal.db.backup
  drupal.drush.site_online
end
#after 'deploy', 'drupal:link_filesystem', 'drupal:drush:set_files_paths', 'drupal:drush:updatedb', 'deploy:
drush:site_online'
after 'deploy' do
  drupal.link_filesystem
  drupal.drush.site_offline
  drupal.drush.set_files_paths
  drupal.drush.enabler(enabled_modules)
```

```
  drupal.drush.updatedb
  drupal.drush.disabler(disabled_modules)
  drupal.drush.revert_features
  drupal.drush.set_theme(default_theme)
  drupal.drush.clear_all_caches
  drupal.drush.site_online
end

after 'deploy:rollback' do
  drupal.link_filesystem
  drupal.drush.clear_all_caches
  drupal.drush.site_offline
  drupal.db.reload_previous
  drupal.drush.site_online
end

before 'drupal:db:grab', 'drupal:drush:site_offline', 'drupal:drush:clear_all_caches'
after 'drupal:db:grab', 'drupal:drush:site_online'

before 'drupal:db:install', 'drupal:drush:site_offline'
after 'drupal:db:install', 'drupal:drush:set_files_paths', 'drupal:drush:clear_all_caches', 'drupal:drush:
site_online'

before 'drupal:db:install_latest', 'drupal:drush:site_offline'
after 'drupal:db:install_latest', 'drupal:drush:set_files_paths', 'drupal:drush:clear_all_caches', 'drupal:
drush:site_online'

after 'deploy:setup', 'drupal:site_setup', 'deploy:update', 'drupal:link_filesystem', 'drupal:
site_setup_permissions'

after 'drupal:post_install_configuration' do
  drupal.drush.set_files_paths
  drupal.drush.enabler(enabled_modules)
  drupal.drush.updatedb
  drupal.drush.disabler(disabled_modules)
  drupal.drush.set_theme(default_theme)
end

after 'drupal:fix_undefined_index', 'drupal:drush:clear_all_caches'

namespace :deploy do
  task :start do ; end
  task :stop do ; end
  task :restart, :roles => :app, :except => { :no_release => true } do
    #Place holder for app restart - in Ruby apps this would touch restart.txt.
  end
end

#Drupal application and project specific tasks.
namespace :drupal do

  desc "Real list of modules to enable"
  task :read_enabled_modules do
    contents = File.open("config/enabled_modules.txt", "r") { |file| file.read }
    return contents
  end

  #desc "Perform a Drupal application deploy."
  # see before/after deploy above
  task :default, :roles => :app, :except => { :no_release => true } do
    site_offline
    clear_all_caches
    backupdb
    deploy.default
    link_filesystem
    set_files_paths
    updatedb
    site_online
  end

  desc "Initial remote setup for Drupal"
```

```
  task :site_setup, :roles => :app do
    commands = []
    commands << "mkdir -p #{deploy_to}/backup"
    commands << "mkdir -p #{deploy_to}/cache"
    commands << "mkdir -p #{deploy_to}/files"
    commands << "mkdir -p #{deploy_to}/log"
    commands << "mkdir -p #{deploy_to}/private_files"
    commands << "mkdir -p #{deploy_to}/subsites"
    commands << "mkdir -p #{deploy_to}/tmp"
    run commands.join(' && ') if commands.any?
  end

  #desc "Set permissions for php to write to some directories"
  task :site_setup_permissions, :roles => :app do
    sudo "chown #{php_user} #{deploy_to}/backup"
    sudo "chown #{php_user} #{deploy_to}/cache"
    sudo "chown #{php_user} #{deploy_to}/files"
    sudo "chown #{php_user} #{deploy_to}/private_files"
    sudo "chown #{php_user} #{deploy_to}/tmp"
    sudo "chown #{php_user} #{deploy_to}/settings.php"
  end

  desc "Get the user to run the Drupal install, then do final configuration"
  task :post_install_configuration, :roles => :app do
    servers = find_servers_for_task(current_task)
    server = servers.first
    default_ans = 'yes'
    ans = Capistrano::CLI.ui.ask "Have you run http://#{server}/install.php? [#{default_ans}]"
    ans = default_ans if ans.empty?
    abort "Please install Drupal first" if ans != default_ans
  end

  #desc "This should not be run on its own - so comment out the description.
  # "Recreate the required Drupal symlinks to static directories and clear all caches."
  task :link_filesystem, :roles => :app, :except => { :no_release => true } do
    commands = []
    commands << "if [ ! -f #{deploy_to}/settings.php ]; then cp #{app_path}/sites/default/default.settings.php #
{deploy_to}/settings.php; fi"
    commands << "cat /dev/null > #{app_path}/.htaccess"
    commands << "mkdir -p #{app_path}/sites/default"
    commands << "ln -nfs #{share_path}/settings.php #{app_path}/sites/default/settings.php"
    commands << "ln -nfs #{share_path}/files #{app_path}/sites/default/files"
    commands << "ln -nfs #{share_path}/tmp #{app_path}/sites/default/tmp"
    commands << "ln -nfs #{share_path}/cache #{app_path}/cache"
    commands << "find #{app_path} -type d -print0 | xargs -0 chmod 755"
    commands << "find #{app_path} -type f -print0 | xargs -0 chmod 644"
    run commands.join(' && ') if commands.any?
  end

  namespace :db do

    desc "returns a stage and time stamped backup file name"
    def backup_file_name(stage)
      backup_time = release_name # use capistrano's variable %Y%m%d%H%M%S
      return make_backup_file_name(stage, backup_time)
    end

    def make_backup_file_name(stage, backup_time)
      filename = "backup/#{stage}-snapshot-#{backup_time}.sql"
      return filename
    end

    desc "gets the most recent database backup file with a given prefix"
    def latest_backup(backup_prefix)
      tmax = Time.at 0
      latestfile = nil
      Dir.glob("backup/#{backup_prefix}-snapshot-*") do |filename|
        mt = File.mtime(filename)
        if mt > tmax
          tmax = mt
          latestfile = filename
```

```
      end
    end
    return latestfile
  end

  desc "upload and install database"
  def upload_install(filename)
    puts "uploading #{filename}"
    top.upload("#{filename}", "#{deploy_to}/#{filename}")
    puts "loading drupal with database"
    drush.load_drupal("#{deploy_to}/#{filename}")
  end

  desc "backup the database"
  task :backup, :roles => :app, :except => { :no_release => true } do
    backup_file = backup_file_name(stage)
    drush.dump_drupal(backup_file)
    puts "#{backup_file}"
  end

  desc "Grab local copy of remote database"
  # cap production drupal:db:grab
  task :grab, :roles => :app, :except => { :no_release => true } do
    backup_file = backup_file_name(stage)
    drush.dump_drupal(backup_file)
    puts "#{backup_file}"
    download "#{deploy_to}/#{backup_file}", "#{backup_file}"
  end

  desc "Install latest local database snapshot"
  # cap staging drupal:db:install_latest -s from=production
  task :install_latest, :roles => :app, :except => { :no_release => true } do
    set :source_stage, fetch(:from, '')
    if source_stage.empty?
      puts "useage:"
      abort "cap staging drupal:db:install_latest -s from=production"
    else
       filename = latest_backup(source_stage)
      if filename.nil?
        abort "first grab a copy of the datatbase: cap #{source_stage} drupal:db:grab"
      else
        puts "Moving database from #{source_stage} to #{stage}"
        upload_install(filename)
      end
    end
  end

  desc "Install a specific local database snapshot"
  # cap staging drupal:db:install -s file=backup/staging-snapshot-2013-10-25-13-17-23.sql
  task :install, :roles => :app, :except => { :no_release => true } do
    set :filename, fetch(:file, '')
    if filename.empty?
      puts "useage:"
      abort "cap staging drupal:db:install -s file=backup/production-snapshot-2013-10-25-13-17-23.sql"
    else
      puts "Moving database from #{filename} to #{stage}"
      upload_install(filename)
    end
  end

  desc "reload database from previous release"
  task :reload_previous, :roles => :app, :except => { :no_release => true } do
    previous_release_name = File.basename(previous_release)
    filename = make_backup_file_name(stage, previous_release_name)
    puts "reloading database from #{filename}"
    drush.load_drupal("#{deploy_to}/#{filename}")
  end

end

namespace :drush do
```

```
desc "Backup the database."
task :backupdb, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} sql-dump --result-file=#{deploy_to}/backup/release-drupal-db.sql"
end

#desc "Backup the database"
# we don't want to require backup_migrate
task :backupdb_bam, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} bam-backup"
end

desc "Clear all caches"
task :clear_all_caches, :roles => :app, :except => { :no_release => true } do
  run "#{drush_cmd} -r #{app_path} --uri=#{drush_uri} cc all"
end

desc "error message fix - undefined index: <name, version>"
# see https://drupal.org/node/1170362
task :fix_undefined_index, :roles => :app, :except => { :no_release => true } do
  run "#{drush_cmd} -r #{app_path} vset --yes install_profile \"standard\""
  run "#{drush_cmd} -r #{app_path} sql-query \"UPDATE system SET status=1 WHERE name='standard';\""
end

desc "Revert all features to install the newest versions"
task :revert_features, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} features-revert-all -y"
end

desc "set the private_files & tmp path for the deploy environment"
task :set_files_paths, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} vset --yes file_public_path sites/default/files"
  run "#{drush_cmd} -r #{app_path} vset --yes file_private_path #{deploy_to}/private_files"
  run "#{drush_cmd} -r #{app_path} vset --yes file_temporary_path sites/default/tmp"
end

desc "Set the site offline"
task :site_offline, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} vset site_offline 1 -y"
  run "#{drush_cmd} -r #{app_path} vset maintenance_mode 1 -y"
end

desc "Set the site online"
task :site_online, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} vset site_offline 0 -y"
  run "#{drush_cmd} -r #{app_path} vset maintenance_mode 0 -y"
end

desc "Run Drupal database updates for new core/modules/themes if required."
task :updatedb, :on_error => :continue do
  run "#{drush_cmd} -r #{app_path} updatedb -y"
end

desc "backup the Drupal database to a timestamped file"
def dump_drupal(filename)
  run "#{drush_cmd} -r #{app_path} sql-dump --result-file=#{deploy_to}/#{filename}"
end

desc "load the Drupal database from an sql dump"
def load_drupal(filename)
  run "`#{drush_cmd} -r #{app_path} sql-connect` < #{filename}"
end

desc "enable an array of modules"
def enabler(modules)
   run "#{drush_cmd} -r #{app_path} pm-enable -y #{modules.join(' ')}"
end

desc "disable an array of modules"
def disabler(modules)
   run "#{drush_cmd} -r #{app_path} pm-disable -y #{modules.join(' ')}"
```

```
    end

    desc "set the default theme"
    def set_theme(theme)
      run "#{drush_cmd} -r #{app_path} vset theme_default #{theme}"
    end

  end

end


namespace :git do

  desc "Place release tag into Git and push it to origin server."
  task :push_deploy_tag do
    user = `git config --get user.name`
    email = `git config --get user.email`
    tag = "release_#{release_name}"
    if exists?(:stage)
      tag = "#{stage}_#{tag}"
    end
    puts `git tag #{tag} #{revision} -m "Deployed by #{user} <#{email}>"`
    puts `git push origin tag #{tag}`
  end

end

namespace :explore do
  desc "Local variables"
  task :locals, :on_error => :continue do
    local_variables.each {|var| puts "#{var} fetch(:#{var})"}
    puts "shared_path #{shared_path}"
    puts "stage #{stage}"
    puts "args #{args}"
    puts "release_name #{release_name}"
    puts "previous_revision #{previous_revision}"
    puts "current_revision #{current_revision}"
    # deploy.instance_variables.map do |var|
    #   puts [var, deploy.instance_variable_get(var)].join(":")
    # end
  end

  desc "check paths for mysql and mysqldump"
  task :check_mysql, :on_error => :continue do
    puts "$PATH"
    run "echo $PATH"
    puts "Which mysql?"
    run "which mysql"
    puts "Which mysqldump?"
    run "which mysqldump"
    puts "connect statement:"
    run "#{drush_cmd} -r #{app_path} sql-connect"
  end

  desc "test du jour"
  task :testdujour, :on_error => :continue do
    mysqlcmd = %x[ echo `#{drush_cmd} -r #{app_path} sql-connect` ]
    puts "mysql command: #{mysqlcmd}"
    puts "current_path #{current_path}"
    puts "deploy_to #{deploy_to}"
  end

  desc "find code for a task"
  task :find, :on_error => :continue do
    # load all the tasks associated with the rails app
    Rails.application.load_tasks

    # get the source locations of actions called by a task
    task_name = 'deploy:cold' # fully scoped task name
    Rake.application[task_name].actions.map(&:source_location)
```

```
  end

  desc "try out enabler"
  task :tryout do
    modules = ['aaa', 'bbb', 'ccc', 'ploop']
    drupal.drush.enabler(modules)
  end

end
```