Drupal Drush Guide

Credits @ Drupal.org

1.1 USAGE

Drush can be run in your shell by typing "drush" from within any Drupal root directory.

```
  $ drush [options] <command> [argument1] [argument2]
```

Use the 'help' command to get a list of available options and commands:

```
  $ drush help
```

For even more documentation, use the 'topic' command:

```
  $ drush topic
```

For a full list of Drush commands and documentation by version, visit
http://drush.ws.

Many commands support a --pipe option which returns machine readable output.
For example, return a list of enabled modules:

```
  $ drush pm-list --type=module --status=enabled --pipe
```

For multisite installations, use the -l option to target a particular site. If you are outside the Drupal web root, you might need to use the -r, -l or other command line options just for Drush to work. If you do not specify a URI with -l and Drush falls back to the default site configuration, Drupal's $GLOBAL['base_url'] will be set to http://default. This may cause some functionality to not work as expected.

```
  $ drush -l http://example.com pm-update
```

Related Options:

```
  -r <path>, --root=<path>      Drupal root directory to use
                         (defaults to current directory or
anywhere in a Drupal directory                                        tree)
  -l <uri> , --uri=<uri>        URI of the Drupal site to use
  -v, --verbose                 Display verbose output.
```

Very intensive scripts can exhaust your available PHP memory. One remedy is to just restart automatically using bash. For example:

```
  while true; do drush search-index; sleep 5; done
```

1.2 DRUSH CONFIGURATION FILES

Inside /path/to/drush/examples you will find some example files to help you get started with your Drush configuration file (example.drushrc.php), site alias definitions (example.aliases.drushrc.php) and Drush commands (sandwich.drush.inc). You will also see an example 'policy' file which can be customized to block certain commands or arguments as required by your organization's needs.

1.3 DRUSHRC.PHP

If you get tired of typing options all the time you can contain them in a drushrc.php file. Multiple Drush configuration files can provide the flexibility of providing specific options in different site directories of a multi-site installation. See example.drushrc.php for examples and installation details.

1.4 SITE ALIASES

Drush lets you run commands on a remote server, or even on a set of remote servers. Once defined, aliases can be references with the @ nomenclature, i.e.

```
  # Syncronize staging files to production
  $ drush rsync @staging:%files/ @live:%files

  # Syncronize database from production to dev, excluding the cache table
  $ drush sql-sync --structure-tables-key=custom --no-cache @live @dev
```

See http://drupal.org/node/670460 and example.aliases.drushrc.php for more information.

1.5 COMMANDS

Drush can be extended to run your own commands. Writing a Drush command is no harder than writing simple Drupal modules, since they both follow the same structure.

See examples/sandwich.drush.inc for light details on the internals of a Drush command file. Otherwise, the core commands in Drush are good models for your own commands.

You can put your Drush command file in a number of places:

a) In a folder specified with the --include option (see `drush topic docs-configuration`).

b) Along with one of your enabled modules. If your command is related to an existing module, this is the preferred approach.

c) In a .drush folder in your HOME folder. Note, that you have to create the .drush folder yourself.

d) In the system-wide Drush commands folder, e.g. /usr/share/drush/commands.

e) In Drupal's sites/all/drush folder. Note, that you have to create the drush folder yourself.

In any case, it is important that you end the filename with ".drush.inc", so that Drush can find it.

1.6 DRUPAL DRUSH COMMAND CHEAT SHEET

When available, there is a shorter version of the same command in parentheses.

cache clear (cc) Clear all caches.

cron Run all cron hooks.

disable (dis) Disable one or more modules.

download (dl) Download core Drupal and projects like CCK, Zen, etc.

enable (en) Enable one or more modules.

eval Evaluate arbitrary php code after bootstrapping Drupal.

help Print this help message. Use --filter to limit command list to one command file (e.g. --filter=pm)

info Release information for a project

installcore (ic) Install Drupal core via the specified install profile.

refresh (rf) Refresh update status information

script Runs the given php script(s) after a full Drupal bootstrap. NOTE: you can't supply absolute paths to the script e.g. ~/Desktop/script.php won't work Desktop/script.php will

sql cli (sqlc) Open a SQL command-line interface using Drupal's credentials.

sql conf Print database connection details.

sql connect A string for connecting to the DB.

sql dump Exports the Drupal DB as SQL using mysqldump.

sql load Copy source database to target database.

sql query (sqlq) Execute a query against the site database.

status (st) Provides a birds-eye view of the current Drupal installation, if any.

statusmodules (sm) Show module enabled/disabled status

sync Rsync the Drupal tree to/from another server using ssh.

test clean Delete leftover tables and files from prior test runs.

test mail Run all tests and mail the results to your team.

uninstall Uninstall one or more modules.

update (up) Update your project code and apply any database updates required (update.php)

updatecode (upc) Update your project code. Moves existing project files to the backup directory specified in the config.

updatedb (updb) Execute the update.php process from the command line.

variable delete (vdel) Delete a variable.

variable get (vget) Get a list of some or all site variables and values.

variable set (vset) Set a variable.

watchdog delete (wd) Delete all messages or only those of a specified type.

watchdog show (ws) Shows recent watchdog log messages. Optionally filter for a specific type.

1.7 EXAMPLES

Drush provides a number of commands that permit you do perform drupal installation, maintenance, and status operations.

1.View the update status of modules

drush -n pm-update

2.Update site modules

drush pm-update

3.Clearing caches

drush cache-clear all OR drush cc

4.Download drush modules

drush pm-download *module1 module2 module3*

5.Enabling modules

drush pm-enable *module1 module2 module3*

6.Disabling modules

drush pm-disable *module1 module2 module3*

7.Download drush to create a new installation

drush pm-download –drupal-project-rename=*my.sitename* drupal

8.Get the list of enabled (disabled) modules

drush pm-list --type=module --status=enabled

9.Get the list of and status of a specific module

drush pm-list --type=module --package="*Package Name*" (if you know the exact package name) ordrush pm-list --type=module | grep '*part of package name or module name*' (if you are not sure of a package or module name)

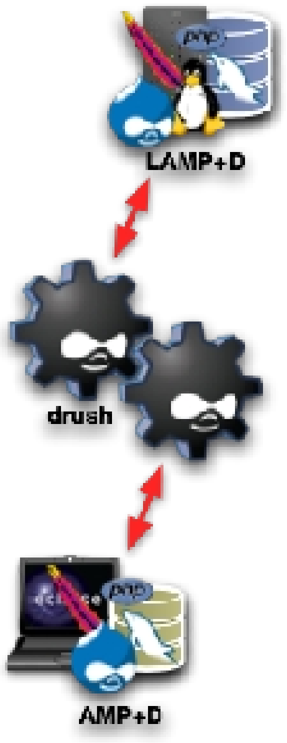
The naming structure relates to file and function naming. The aliases for the above are:

•pm-update: up
•cache-clear: cc
•pm-download: dl
•pm-enable: en
•pm-disable: dis
•pm-list: sm

1.8 USING DRUSH TO SYNCHRONIZE AND DEPLOY SITES

Drush has introduced the concept of 'site aliases', which are small arrays containing connection details relevant to individual site instances. These are used to note information about your local dev site and a remote staging site (for example).

With this info, drush can help you move content between these locations!



After installing drush, have a look at:
drush help site-alias
drush help rsync
drush help sql-sync
to get started.
You will first want to make a note of your local site alias information.

About the local setup

If I am working on a drupal multisite instance called **demosite.drupal6.local**, my working site directory may be something like /var/www/drupal6/sites/demosite.drupal6.local

I will be wanting to connect to a location called **demosite.staging.remote**. These are our two 'site aliases'.
As implied, the aliases can actually be shorter names, like '@dev' and '@stage'. In the context of a Drupal site, you can also use the name of the site as it appears in the 'sites' folder. In order for this to work, you must have -r / --root set in your drushrc.php file, or your current working directory must be inside the Drupal root. If you have done this, you can use the site folder name as a site alias, like so:
drush site-alias --full demosite.drupal6.local

This tells us:

```
$aliases['demosite.drupal6.local'] = array (
  'uri' => 'demosite.drupal6.local',
  'root' => '/var/www/drupal6',
);
```

Pretty simple so far. There is more to it.

Enter information about the remote target peer alias
We also need to have some information about the destination site. The remote site is (of course) not local, so the local drush knows nothing about it until we tell it. As mentioned in the help documentation `drush help rsync`, you can **read 'example.aliases.drushrc.php'** to see an example of how this can be done.

Your own aliases configuration file can be placed in several places. Since the remote site for demosite.drupal6.local is only relevant to this site, I will put it in the local site dir as **`{drupal}/sites/demosite.drupal6.local/peer.alias.drushrc.php`.**

Edit the file (create it if needed)
`{drupal}/sites/demosite.drupal6.local/peer.alias.drushrc.php`

To add the remote sites information:

```
<?php
$aliases['peer'] = array (
  'uri' => 'demosite.staging.remote',
  'root' => '/var/www/vhosts/staging/httpdocs',
  'remote-host' => 'mystagingserver.myisp.com',
  'remote-user' => 'publisher',
);
?>
```

Tip: This is documented much more in the example.aliases.drushrc.phpdistribution.
Tip: Creating the site-alias config array is tedious by hand. If you have a working site, you can just ask it to give you all the details like so:
Change into the site dir of a working site and run
`drush site-alias --with-db --show-password --with-optional @self`

I often go

`drush site-alias --with-db --show-password --with-optional @self > /etc/drush/mysqit.alias.drushrc.php`

and then **importantly** edit the resulting new files and A: add a `<?php` tag to the top! B: relabel it from @self to your preferred nickname - which must match the filename you used.

Those extra connection details are required for remote aliases.

If you want, you can also split out the component parts of an alias and use inheritance to construct the peer alias. For example:

```
<?php
$aliases['mystagingserver'] = array (
  'remote-host' => 'mystagingserver.myisp.com',
  'remote-user' => 'publisher',
);
$aliases['peer'] = array (
  'parent' => '@mystagingserver',
  'uri' => 'demosite.staging.remote',
  'root' => '/var/www/vhosts/staging/httpdocs',
);
?>
```

Check that this information is now available to drush.

```
drush demosite.drupal6.local site-alias

# Fetch a list of known sites

drush demosite.drupal6.local site-alias --full @peer

# Show what we know about the named site
```

... it should reflect back the connection details we've entered. Note that because we placed the '@peer' alias inside the site folder for our demo site, this alias is only known in contexts where that site has been named or bootstrapped by drush. Had we put the alias file inside of a more global location such as $HOME/.drush, then it would be globally available. In that case, we would probably want to use an alias name that is a little less generic than 'peer'.

It may feel strange to put the alias context, demosite.drupal6.local in the previous example, on the left side of the site-alias command. A more direct way to specify the context of the @peer alias would be to use relative alias syntax:

```
drush site-alias --full demosite.drupal6.local/@peer
```

drush rsync the files

The first time we do this, it will create the remote site entirely from scratch (it didn't exist before). This uses rsync under the hood, so it will do only incremental updates and preserve permissions etc if possible.

```
drush rsync demosite.drupal6.local @peer

You will destroy data from
 publisher@mystagingserver.myisp.com:/var/www/vhosts/staging/httpdocs/ and replace with data from /var/www/drupal6/
```

> **Requirement:** For drush to communicate with remote servers, **you must first set up ssh authentication keys** to assist automated logins. If you haven't already, go and look into this. It's pretty easy once you know how, and a huge win for productivity. greg.1.anderson has supplied a script that may help this set-up. It only has to be done once per user account per server.

Cool. The remote site has all the current files up there now!

> **Gotcha:** This command put up everything, even the unrelated multisites I had hanging around. There are probably tidier ways to do this.

Aside: a fix for multisite sites folder aliases

I'm not using the sites/default directory, which means my local and remote hosts will have a slightly different expectation about the sitename I will use for the site instance. This can cause trouble, but a quick, helpful way to deal with this is to just symlink the site directory with the alternative name.
On the remote site:

```
cd /var/www/vhosts/staging/httpdocs/sites;
ln -s demosite.drupal6.local demosite.staging.remote
```

... now the remote site will serve the same content and as the local one. *There are other ways to work around this issue*, and this approach can have side-effects, but that won't be covered here.

Sync the database

We'll also need to push up the database. It can be done in one line if everything is already set up at the other end.

The remote site will not quite use the same settings, because we'll need different database settings etc. Cleverly, the rsync has carefully chosen to `--exclude="settings.php"`
(see "drush help rsync")

This means we get to define the remote DB settings separately, as required.
If you've not already done so, set up your database however you normally would on the remote site. This is different between hosts, so probably not automated.

Quickest is to [CREATE DATABASE; GRANT PRIVILEGES; Copy and change the settings.php by hand to edit the $db_url], although you can probably even run the Drupal install wizard if you like.

With the remote database ready to take the content, we'll push it up. This will over-write anything that used to be there.

```
drush sql-sync demosite.drupal6.local @peer
```

This command will actually log in to the remote site, and investigate the settings found there to find out what the remote database connection string is. Pretty nifty. You can do other things remotely using this channel also.

This is easiest if drush is also available on the remote site.

If you've not already set this up, you can use drush to push itself to a remote server: drush rsync demosite.drupal6.local:%drush @peer:%drush.
You can avoid installing drush remotely by noting the database connection details locally in the site-aliases array.

> **Gotcha:** Depending on how you installed it, the commandline install of the drush program may not be found on the remote site when running a non-interactive shell session. This can be frustrating.
> To work around this, the alias file allows you to define `path-aliases['%drush-script'] = '/path/to/drush';`.

See example.aliases.drushrc.php. **NOTE** the parameter '%drush-script' must link to the drush shell script file or the parameter '%drush' must link to the drushdirectory. Either works.

Tip: Pay attention to the advice in example.aliases.drushrc.php on creating a designated '!dump' file path. Note that `%dump` is a file name, not a folder, and the directory structure to it must already exist.
for mine, I set it to:
`'%dump' => '/var/www/vhosts/staging/data/dump.sql',`

Gotcha: `drush sql-sync` is **not yet aware of database prefixes** and may transfer a heck of a lot more than you wanted.

Done

If you got this far, and worked through any troubleshooting issues, you should now have a working remote clone of your site.
From now on it will be MUCH easier, and you can sync file and DB up and down, selectively or in bulk.

EG, to push up some recent changes to a theme from dev to staging

```
drush  rsync demosite.drupal6.local/sites/all/themes/demo1/
@peer:sites/all/themes/demo1/
```

to fetch the latest files that may have been uploaded to the staging site

```
drush rsync @peer:%files/ demosite.drupal6.local:%files
```

It probably be a good idea to flush the site caches *before* transferring the db from local, or on the remote site *after* a db transfer. drush provides a shortcut for that.

```
drush cc all
```

(run on either local or remote system, respectively)

(there are probably more shortcuts also)

Managing remote sites with drush aliases

Once you've got remote aliases set up, you can run drush commands on them directly from your local commandline. Drush will handle to logins and relative paths needed.

```
drush @peer status
```

will

- Check for an alias entry named 'peer'
- Log in to the remote-site using the given credentials
- Move to the appropriate directory, or at least use the correct remote paths
- invoke drush there to carry out the command
- and return the result

This should work for most drush actions, just by putting an @alias identifier before the commands. Internally, this triggers the functions `drush_remote_command()` `drush_do_site_command()` and `drush_backend_invoke()`

Commands like

drush @peer update
drush @peer cc

Will run DB updates or clear the cache on a *remote* site without you ever leaving the comfort of your local shell.

To work fully, the remote site must have drush installed also. See the note above about `path-aliases['%drush-script'] = '/path/to/drush';` if you are having trouble getting that to work.