Command Line

The following are some of the more commonly used commands in the `meteor` command-line tool. This is just an overview and does not mention every command or every option to every command; for more details, use the `meteor help` command.

meteor help

Get help on meteor command line usage. Running `meteor help` by itself will list the common meteor commands. Running meteor help command will print detailed help about the command.

meteor run

Run a meteor development server in the current project. Searches upward from the current directory for the root directory of a Meteor project. Whenever you change any of the application's source files, the changes are automatically detected and applied to the running application.

You can use the application by pointing your web browser at localhost:3000. No Internet connection is required.

This is the default command. Simply running `meteor` is the same as `meteor run`.

To pass additional options to Node.js use the `SERVER_NODE_OPTIONS` environment variable. E.g. for Windows PowerShell: `$env:SERVER_NODE_OPTIONS = '--inspect' | meteor run`. Or for Linux: `SERVER_NODE_OPTIONS=--inspect-brk meteor run`.

To specify a port to listen on (instead of the default 3000), use `--port [PORT]`. (The development server also uses port `N+1` for the default MongoDB instance)

For example: `meteor run --port 4000` will run the development server on `http://localhost:4000` and the development MongoDB instance on `mongodb://localhost:4001`.

Run `meteor help run` to see the full list of options.

meteor debug

Run the project, but suspend the server process for debugging.

> **NOTE:** The `meteor debug` command has been superseded by the more flexible `--inspect` and `--inspect-brk` command-line flags, which work for any `run`, `test`, or `test-packages` command.
>
> The syntax of these flags is the same as the equivalent Node.js flags, with two notable differences:
>
> - The flags affect the server process spawned by the build process, rather than affecting the build process itself.
> - The `--inspect-brk` flag causes the server process to pause just after server code has loaded but before it begins to execute, giving the developer a chance to set breakpoints in server code.

The server process will be suspended just before the first statement of server code that would normally execute. In order to continue execution of server code, use either the web-based Node Inspector or the command-line debugger (further instructions will be printed in the console).

Breakpoints can be set using the `debugger` keyword, or through the web UI of Node Inspector ("Sources" tab).

The server process debugger will listen for incoming connections from debugging clients, such as node-inspector, on port 5858 by default. To specify a different port use the `--debug-port <port>` option.

The same debugging functionality can be achieved by adding the `--debug-port <port>` option to other `meteor` tool commands, such as `meteor run` and `meteor test-packages`.

> **Note:** Due to a bug in `node-inspector`, pushing "Enter" after a command on the Node Inspector Console will not successfully send the command to the server. If you require this functionality, please consider using Safari or `meteor shell` in order to interact with the server console until the `node-inspector` project fixes the bug. Alternatively, there is a hot-patch available in this comment on #7991.

meteor create name

Create a new Meteor project. By default, it uses React and makes a subdirectory named *name* and copies in the template app. You can pass an absolute or relative path.

Flags

Flags for default packages

`--bare`

Creates a basic, blaze project.

`--full`

Creates a more complete, imports-based project which closely matches the file structure recommended by the Meteor Guide

`--minimal`

Creates a project with as few Meteor Packages as possible.

`--package`

Creates a new package. If used in an existing app, this command will create a package in the packages directory.

`--typescript`

Create a basic Typescript React-based app. Can be combined with other flags to use a different UI than React.

`--apollo`

Create a basic Apollo + Reac app.

Flags for default UI libraries / frameworks

`--blaze`

`--vue`

Create a basic vue-based app. See the Vue guide for more information.

`--svelte`

Create a basic Svelte app.

Packages

	Default (`--react`)	`--bare`	`--full`	`--minimal`	`--blaze`	`--apollo`	`--vue`	`--svelte`
autopublish	X				X			X
akryum:vue-component							X	
apollo						X		
blaze-html-templates			X		X			
ecmascript	X	X	X	X	X	X	X	X
es5-shim	X	X	X	X	X	X	X	X
hot-module-replacement	X				X	X		
insecure	X				X			X
johanbrook:publication-collector			X			X		

Default (`--react`)	`--bare`	`--full`	`--minimal`	`--blaze`	`--apollo`	`--vue`	`--svelte`
jquery		X		X			
ostrio:flow-router-extra		X					
less		X					
meteor			X				
meteor-base X	X	X		X	X	X	X
mobile-experience X	X	X		X	X	X	X
mongo X	X	X		X	X	X	X
meteortesting:mocha		X				X	
reactive-var X	X	X		X	X	X	X
rdb:svelte-meteor-data							X
server-render			X		X	X	
shell-server	X		X	X	X	X	X
standard-minifier-css X	X	X	X	X	X	X	X
standard-minifier-js X	X	X	X	X	X	X	X
static-html	X		X		X	X	X
svelte:compiler							X
swydo:graphql					X		
tracker	X	X		X		X	
typescript X	X	X	X	X	X	X	X
webapp			X				
react-meteor-data X							

meteor login / logout

Log in and out of your account using Meteor's authentication system.

You can pass `METEOR_SESSION_FILE=token.json` before `meteor login` to gen-

erate a login session token so you don't have to share your login credentials with third-party service providers.

Once you have your account you can log in and log out from the command line, and check your username with `meteor whoami`.

meteor deploy site

Deploy the project in your current directory to Galaxy.

Use `--owner` to decide which organization or user account you'd like to deploy a new app to if you are a member of more than one Galaxy-enabled account.

You can deploy in debug mode by passing `--debug`. This will leave your source code readable by your favorite in-browser debugger, just like it is in local development mode.

To delete an application you've deployed, specify the `--delete` option along with the site.

You can add information specific to a particular deployment of your application by using the `--settings` option. The argument to `--settings` is a file containing any JSON string. The object in your settings file will appear on the server side of your application in `Meteor.settings`.

Settings are persistent. When you redeploy your app, the old value will be preserved unless you explicitly pass new settings using the `--settings` option. To unset `Meteor.settings`, pass an empty settings file.

{% pullquote warning %} `free` and `mongo` options were introduced in Meteor 2.0 {% endpullquote %}

You can run your app for free using the option `--free`. But, there are some limitations. The first one is that you cannot use a custom domain to run a free app. Your domain must contain a Meteor domain name (`.meteorapp.com` to US region, `.au.meteorapp.com` to Asia region, or `.eu.meteorapp.com` to Europe region). Second thing you must know is that your free apps have Cold Start enabled. Cold Start means that your app will stop if it has no connection for 10 minutes, and it will go automatically up when someone tries to connect to it. The third thing you must know is that free apps run on one, and just one, Tiny container. This is important to know, because Tiny containers are NOT meant to production environment, so even small apps can crash with a lot of connections. To keep your app on free, you always need to provide this option.

With the option `--mongo` you can deploy your app without having to pay for a MongoDB provider. By providing this option, Galaxy will create a database for you in our shared cluster and inject the mongo URL on your settings. So with this, you don't even need to provide the settings file anymore (if your settings files just have the mongo URL of course). This is great to test apps, but it shouldn't be used in a production environment, as you will be running in a shared Cluster with limited space. The rules behind this option are: If it is the

first deploy of the app, and you provided the option `--mongo`, after the deploy is finished you will receive your mongo URL on your console (you can also see your URL on Galaxy in your app's version). You can put that URL on your settings file if want to. If you try to do a second without the option `--mongo` and without providing a mongo URL on your settings, your deploy will fail as usual. If you provide the option `--mongo` and a mongo URL, the mongo URL on your settings file is the one that will be used by Galaxy to connect your app to a MongoDB. One last thing, you need to have at least one document in your database so Meteor is really going to instantiate it. Then you will be able to access it using any MongoDB client with the provided URI.

Use the options `--mongo` and `--free` to easily deploy a free app already with a mongo database connected to it.

{% pullquote warning %} Free apps and MongoDB shared hosting: Meteor Software reserves the right to stop or remove applications we deem to be abusing the free plan offering at any time. Please be advised that the free plan offering is not recommended for production applications. The shared MongoDB cluster that comes configured with the free plan does not provide backups or restoration resources. {% endpullquote %}

{% pullquote warning %} If you want to connect to your free MongoDB shared cluster using your on settings make sure you include this option in your settings in the Mongo package configuration section:

```
packages: {
  mongo: {
    options: {
        tlsAllowInvalidCertificates: true,
    },
  },
}
```

This is necessary as our database provider does not have certificates installed on every machine and we don't want to force apps to have this certificate. More about this option here {% endpullquote %}

You can change the app plan by providing argument `--plan` with one of the following values: professional, essentials, or free. Be aware that this argument overwrites the `--free` argument.

{% pullquote warning %} The `plan` option is available since Meteor 2.1. {% endpullquote %}

Use `--cache-build` to keep the bundle in your temp folder after the deploy is finished, this is helpful when you want to deploy the same code to different environments. For example, a background job app from the same code as the web app.

Your project should be a git repository as the commit hash is going to be used

to decide if your code is still the same or not in the next deploy.

{% pullquote warning %} The `cache-build` option is available since Meteor 1.11. {% endpullquote %}

With the argument `--container-size` you can change your app's container size using the deploy command. The valid arguments are: `tiny`, `compact`, `standard`, `double`, `quad`, `octa`, and `dozen`. To see more about the difference and prices of each one you can check it here.

{% pullquote warning %} The `--container-size` option is available since Meteor 2.4.1. {% endpullquote %}

meteor update

Attempts to bring you to the latest version of Meteor, and then to upgrade your packages to their latest versions. By default, update will not break compatibility.

For example, let's say packages A and B both depend on version 1.1.0 of package X. If a new version of A depends on X@2.0.0, but there is no new version of package B, running `meteor update` will not update A, because doing so will break package B.

You can pass in the flag `--packages-only` to update only the packages, and not the release itself. Similarly, you can pass in names of packages (`meteor update foo:kittens baz:cats`) to only update specific packages.

Every project is pinned to a specific release of Meteor. You can temporarily try using your package with another release by passing the `--release` option to any command; `meteor update` changes the pinned release.

Sometimes, Meteor will ask you to run `meteor update --patch`. Patch releases are special releases that contain only very minor changes (usually crucial bug fixes) from previous releases. We highly recommend that you always run `update --patch` when prompted.

You may also pass the `--release` flag to act as an override to update to a specific release. This is an override: if it cannot find compatible versions of packages, it will log a warning, but perform the update anyway. This will only change your package versions if necessary.

meteor add package

Add packages to your Meteor project. By convention, names of community packages include the name of the maintainer. For example: `meteor add iron:router`. You can add multiple packages with one command.

Optionally, adds version constraints. Running `meteor add package@1.1.0` will add the package at version `1.1.0` or higher (but not `2.0.0` or higher). If you want to use version `1.1.0` exactly, use `meteor add package@=1.1.0`. You can also 'or' constraints together: for example, `meteor add 'package@=1.0.0 || =2.0.1'` means either 1.0.0 (exactly) or 2.0.1 (exactly).

To remove a version constraint for a specific package, run `meteor add` again without specifying a version. For example above, to stop using version `1.1.0` exactly, run `meteor add package`.

meteor remove package

Removes a package previously added to your Meteor project. For a list of the packages that your application is currently using, run `meteor list`.

This removes the package entirely. To continue using the package, but remove its version constraint, use `meteor add`.

Meteor does not downgrade transitive dependencies unless it's necessary. This means that if running `meteor add A` upgrades A's parent package X to a new version, your project will continue to use X at the new version even after you run `meteor remove A`.

meteor list

Lists all the packages that you have added to your project. For each package, lists the version that you are using. Lets you know if a newer version of that package is available.

Flags

Flags are optional and can be used to format the output. The default output requires no flags whatsoever. The following flags are supported:

`--tree`

Outputs a tree showing how packages are referenced.

`--json`

Outputs an unformatted JSON String, showing how packages are referenced.

`--weak`

Show weakly referenced dependencies in the tree. Only functional in combination with `--tree` or `--json`.

`--details`

Adds more package details to the JSON output. Only functional in combination with `--json`.

meteor add-platform platform

Adds platforms to your Meteor project. You can add multiple platforms with one command. Once a platform has been added, you can use 'meteor run platform' to run on the platform, and `meteor build` to build the Meteor project for every added platform.

meteor remove-platform platform

Removes a platform previously added to your Meteor project. For a list of the platforms that your application is currently using, see `meteor list-platforms`.

meteor list-platforms

Lists all of the platforms that have been explicitly added to your project.

meteor ensure-cordova-dependencies

Check if the dependencies are installed, otherwise install them.

meteor mongo

Open a MongoDB shell on your local development database, so that you can view or manipulate it directly.

{% pullquote warning %} For now, you must already have your application running locally with `meteor run`. This will be easier in the future. {% endpullquote %}

meteor reset

Reset the current project to a fresh state. Removes the local mongo database.

{% pullquote warning %} This deletes your data! Make sure you do not have any information you care about in your local mongo database by running `meteor mongo`. From the mongo shell, use `show collections` and db.collection.find() to inspect your data. {% endpullquote %}

{% pullquote warning %} For now, you can not run this while a development server is running. Quit all running meteor applications before running this. {% endpullquote %}

meteor build

Package this project up for deployment. The output is a directory with several build artifacts:

a tarball (.tar.gz) that includes everything necessary to run the application server (see the README in the tarball for details). Using the `--directory` option will produce a `bundle` directory instead of the tarball.

an unsigned apk bundle and a project source if Android is targeted as a mobile platform

a directory with an Xcode project source if iOS is targeted as a mobile platform

You can use the application server bundle to host a Meteor application on your own server, instead of deploying to Galaxy. You will have to deal with logging, monitoring, backups, load-balancing, etc, all of which we handle for you if you use Galaxy.

The unsigned `apk` bundle and the outputted Xcode project can be used to deploy your mobile apps to Android Play Store and Apple App Store.

By default, your application is bundled for your current architecture. This may cause difficulties if your app contains binary code due to, for example, npm packages. You can try to override that behavior with the `--architecture` flag.

You can set optional data for the initial value of `Meteor.settings` in your mobile application with the `--mobile-settings` flag. A new value for `Meteor.settings` can be set later by the server as part of hot code push.

You can also specify which platforms you want to build with the `--platforms` flag. Examples: `--platforms=android`, `--platforms=ios`, `--platforms=web.browser`.

meteor lint

Run through the whole build process for the app and run all linters the app uses. Outputs all build errors or linting warnings to the standard output.

meteor search

Searches for Meteor packages and releases, whose names contain the specified regular expression.

meteor show

Shows more information about a specific package or release: name, summary, the usernames of its maintainers, and, if specified, its homepage and git URL.

meteor publish

Publishes your package. To publish, you must `cd` into the package directory, log in with your Meteor Developer Account and run `meteor publish`. By convention, published package names must begin with the maintainer's Meteor Developer Account username and a colon, like so: `iron:router`.

To publish a package for the first time, use `meteor publish --create`.

Sometimes packages may contain binary code specific to an architecture (for example, they may use an npm package). In that case, running publish will only upload the build to the architecture that you were using to publish it. You can use `publish-for-arch` to upload a build to a different architecture from a different machine.

If you have already published a package but need to update it's metadata (the content of `Package.describe`) or the README you can actually achieve this via `meteor publish --update`.

meteor publish-for-arch

Publishes a build of an existing package version from a different architecture.

Some packages contain code specific to an architecture. Running `publish` by itself, will upload the build to the architecture that you were using to publish. You need to run `publish-for-arch` from a different architecture to upload a different build.

For example, let's say you published name:cool-binary-blob from a Mac. If you want people to be able to use cool-binary-blob from Linux, you should log into a Linux machine and then run `meteor publish-for-arch name:cool-binary-blob@version`. It will notice that you are on a linux machine, and that there is no Linux-compatible build for your package and publish one.

Currently, the supported architectures for Meteor are 32-bit Linux, 64-bit Linux and Mac OS. Galaxy's servers run 64-bit Linux.

meteor publish-release

Publishes a release of Meteor. Takes in a JSON configuration file.

Meteor releases are divided into tracks. While only MDG members can publish to the default Meteor track, anyone can create a track of their own and publish to it. Running `meteor update` without specifying the `--release` option will not cause the user to switch tracks.

To publish to a release track for the first time, use the `--create-track` flag.

The JSON configuration file must contain the name of the release track (`track`), the release version (`version`), various metadata, the packages specified by the release as mapped to versions (`packages`), and the package & version of the Meteor command-line tool (`tool`). Note that this means that forks of the meteor tool can be published as packages and people can use them by switching to a corresponding release. For more information, run `meteor help publish-release`.

meteor test-packages

Test Meteor packages, either by name, or by directory. Not specifying an argument will run tests for all local packages. The results are displayed in an app that runs at `localhost:3000` by default. If you need to, you can pass the `--settings` and `--port` arguments.

meteor admin

Catch-all for miscellaneous commands that require authorization to use.

Some example uses of `meteor admin` include adding and removing package maintainers and setting a homepage for a package. It also includes various helpful functions for managing a Meteor release. Run `meteor help admin` for more information.

meteor shell

When `meteor shell` is executed in an application directory where a server is already running, it connects to the server and starts an interactive shell for evaluating server-side code.

Multiple shells can be attached to the same server. If no server is currently available, `meteor shell` will keep trying to connect until it succeeds.

Exiting the shell does not terminate the server. If the server restarts because a change was made in server code, or a fatal exception was encountered, the shell will restart along with the server. This behavior can be simulated by typing `.reload` in the shell.

The shell supports tab completion for global variables like `Meteor`, `Mongo`, and `Package`. Try typing `Meteor.is` and then pressing tab.

The shell maintains a persistent history across sessions. Previously-run commands can be accessed by pressing the up arrow.

meteor npm

The `meteor npm` command calls the `npm` version bundled with Meteor itself.

Additional parameters can be passed in the same way as the `npm` command (e.g. `meteor npm rebuild`, `meteor npm ls`, etc.) and the npm documentation should be consulted for the full list of commands and for a better understanding of their usage.

For example, executing `meteor npm install lodash --save` would install `lodash` from npm to your `node_modules` directory and save its usage in your `package.json` file.

Using the `meteor npm ...` commands in place of traditional `npm ...` commands is particularly important when using Node.js modules that have binary dependencies that make native C calls (like `bcrypt`) because doing so ensures that they are built using the same libaries.

Additionally, this access to the npm that comes with Meteor avoids the need to download and install npm separately.

meteor node

The `meteor node` command calls the `node` version bundled with Meteor itself.

> This is not to be confused with `meteor shell`, which provides an almost identical experience but also gives you access to the "server" context of a Meteor application. Typically, `meteor shell` will be preferred.

Additional parameters can be passed in the same way as the `node` command, and the Node.js documentation should be consulted for the full list of commands and for a better understanding of their usage.

For example, executing `meteor node` will enter the Node.js Read-Eval-Print-Loop (REPL) interface and allow you to interactively run JavaScript and see the results.

Executing `meteor node -e "console.log(process.versions)"` would run `console.log(process.versions)` in the version of `node` bundled with Meteor.