# Tasks

The next important feature in Visual Studio Code is tasks. Thanks to tasks you can run any command line tool directly from Visual Studio Code and get output there. So, you can run any compilers, deploying tools, packaging tools and so on.

In order to create a task you need to start with **tasks.json** file that can contain one or more task definitions.

Let’s start with the simplest task that doesn’t have any sense but you can create it quickly and understand some principles there. I propose to create the following tasks.json file:

*{*

*"version": "0.1.0",*

*"command": "git",*

*"isShellCommand": true,*

*"args": [*

*],*

*"tasks": [*

*{*

*"taskName": "gitinfo",*

*"args": ["--version"],*

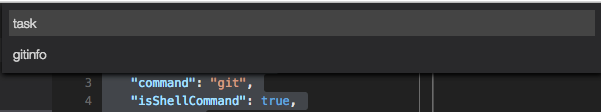
*"isBuildCommand": true*

*}*

*]*

*}*

This task allows you to run **git** tool with **version** parameter. To run this task, you can click ⇧⌘B combination or use ⇧⌘P to run Command Palette window where you can type Run Task command. When running this command, Visual Studio Code will display a list of all available tasks and you can select this one:



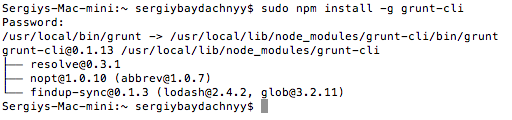
Using this task, you can easily understand some of the elements of **tasks.json**. These are the most common attributes:

* **command** – name of the tool itself that you can use in Terminal to activate the tool;
* **isShellCommand** – allows to run the tool directly in Code shell;
* **tasks** – list of tasks;
* **taskName** – name for a task. Code will use this name to allow us to select the right task;
* **args** – arguments of command line;
* **isBuildCommand** – allows to activate the task using ⇧⌘B combination;

Of course, in most cases you will use more complex commands and usually you will use tools like msbuild, gulp, grunt, make, ant and others. These tools allow to use your own configuration file, and all that you need to do is just to create a simple tasks.json file that activates any of these tools. Let’s use grunt tool that is very popular among Node.js developers.

In order to use grunt you need to prepare your development environment and at the first step you need to install Node.js itself. It’s easy to do, visiting the <https://nodejs.org/en/> site. Just download the package and install it.

Once Node.js is installed you can install grunt:



You can see that I use sudo command because grunt installer requires Administrator permissions.

I decided to avoid any complex scenarios but just visited <http://gruntjs.com/getting-started> and copied some code from the tutorial. First of all, we need **package.json** file that contains information about basic plugins which are popular for grunt and description of my project, including .js files there:

*{*

*"name": "mainPrj",*

*"version": "0.1.0",*

*"devDependencies": {*

*"grunt": "~0.4.5",*

*"grunt-contrib-jshint": "~0.10.0",*

*"grunt-contrib-nodeunit": "~0.4.1",*

*"grunt-contrib-uglify": "~0.5.0"*

*},*

*"files": [*

*"main.js"*

*]*

*}*

and I created gruntfile.js itself that contains grunt tasks:

*module.exports = function(grunt) {*

*// Project configuration.*

*grunt.initConfig({*

*pkg: grunt.file.readJSON('package.json'),*

*uglify: {*

*options: {*

*banner: '/\*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> \*/\n'*

*},*

*build: {*

*src: '\*.js',*

*dest: '<%= pkg.name %>.min.js'*

*}*

*}*

*});*

*// Load the plugin that provides the "uglify" task.*

*grunt.loadNpmTasks('grunt-contrib-uglify');*

*// Default task(s).*

*grunt.registerTask('default', ['uglify']);*

*};*

It’s easy to understand that this grunt task will read all js files from main folder and combine them in a single file, removing all not needed symbols.

It’s almost done but before creating the task in Code you need to install all plugins defined in **project.json** file. Just activate Terminal session in the context of this file and use the following command:

**sudo npm install**

So, the environment is set up, and you can create a simple task:

*{*

*"version": "0.1.0",*

*"command": "grunt",*

*"isShellCommand": true*

*}*

If you use Command Palette window and type Run Task you can see that Code recognized the task name inside the grintfile.js, and you can run the task. In output window you can see the result and if everything is OK you can find one more file in the folder with all javascript inside.

Usually I like to use Output window to understand result of a task but it’s possible to use existing problem matchers and even define your own. If you like it, you can refer to the documentation.

So, tasks are a very powerful mechanism that allows to run everything inside Visual Studio Code.