# Задачи

Следующая важная возможность в Code, это задачи. Благодаря задачам Вы можете выполнить любую команду командной строки в контексте Visual Studio Code и просмотреть результаты работы прямо из среды разработки. Таким образом Вы можете использовать внешние компиляторы, отладчики, средства тестирования и другое.

Давайте начнем с простейшей задачи, которая не имеет особого смысла, но на ее примере мы разберем структуру файла с описанием задач. Создайте task.json файл и внесите туда следующий текст:

*{*

*"version": "0.1.0",*

*"command": "git",*

*"isShellCommand": true,*

*"args": [*

*],*

*"tasks": [*

*{*

*"taskName": "gitinfo",*

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

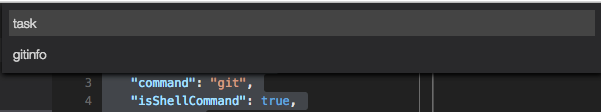
*"isBuildCommand": true*

*}*

*]*

*}*

Эта задача позволяет запустить git утилиту с параметром version. Чтобы запустить эту задачу используйте комбинацию ⇧⌘B или используйте ⇧⌘P для вызова Command Palette. В последнем Вы можете выполнить команду Run Task и получить список доступных команд для выполнения:



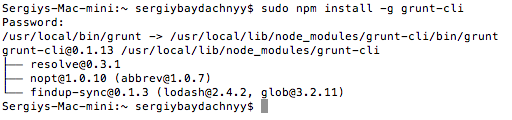
На примере этой задачи мы можем рассмотреть список наиболее общих элементов в task.json файле:

* **command** – тут Вы должны указать имя утилиты, которое Вы можете использовать для ее вызова из терминала;
* **isShellCommand** – позволяет запустить утилиту непосредственно в оболочке Visual Studio Code;
* **tasks** – позволяет задать список задач;
* **taskName** – имя конкретной задачи, которое можно использовать из Command Palette;
* **args** – аргументы командной строки;
* **isBuildCommand** – позволяет активировать задачу с помощью ⇧⌘B комбинации клавиш;

Конечно, в большинстве случаев Вы будете использовать более сложные команды и утилиты. Наиболее популярными утилитами являются msbuild, gulp, grunt, make, ant. Обычно при использовании этих утилит Вы должны определить собственный конфигурационный файл, который понимает выбранная утилита, а затем включить ее вызов в tasks.json.

Давайте попробуем рассмотреть более сложный пример и возьмем утилиту grunt, которая очень популярна среди разработчиков на node.js.

Чтобы использовать grunt вам необходимо подготовить окружение. Во-первых, необходимо установить node.js, чтобы сделать это, достаточно открыть ссылку <https://nodejs.org/en/> и загрузить пакет оттуда. Как только node.js установлен, Вы можете установить grunt:



Для установки мы должны использовать права администратора, поэтому я использовал sudo.

Я попробовал избежать сложных сценариев, поэтому решил ограничиться примером на странице <http://gruntjs.com/getting-started> и просто скопировал пример конфигурации оттуда. Итак, первым делом нам нужно создать package.json файл, который будет содержать информацию о нашем проекте и об используемых расширениях:

*{*

*"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"*

*]*

*}*

В моем случае у меня только один файл в проекте main.js, но в реальности их будет намного больше.

Теперь можно создать файл конфигурации для grunt:

*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']);*

*};*

Этот модуль для grunt позволяет обработать все файлы в каталоге нашего проекта и записать из содержимое в один файл, предварительно удалив все ненужные символы для компилятора.

Мы практически закончили, но теперь нужно установить все расширения, которые мы прописали в package.json. Это можно сделать, открыв терминал в контексте рабочего каталога и запустив следующую команду

**sudo npm install**

Таким образом, окружение полностью настроено и можно приступить к созданию задачи:

*{*

*"version": "0.1.0",*

*"command": "grunt",*

*"isShellCommand": true*

*}*

Открыв Command Palette и набрав Run Task, Вы можете увидеть, что Code распознал все задачи в конфигурационном файле для grunt и их можно запустить.

В результате работы задачи, Вы сможете увидеть новый JavaScript файл со всем исходным кодом внутри, преобразованным по алгоритму. При этом, результаты работы будут отображаться в Output окне, что позволит Вам обнаружить проблему, если таковая есть.

Visual Studio Code поддерживает специальные расширения, которые позволяют обрабатывать данные из output окна и преобразовывать их в форматированные сообщения, отображающиеся в списке ошибок. Данный подход описан в документации.