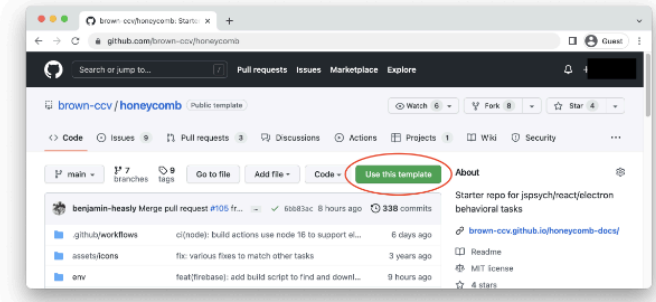
Version: 3.4.x

Quick Start

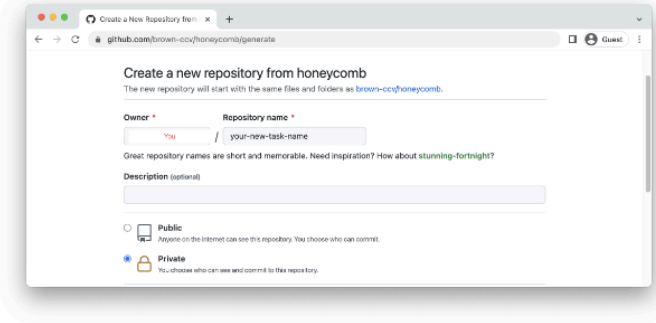
Creating Your Task Repository

The Honeycomb repository is a template and serves as the starting point for all tasks. Creating your repository from the template starts your project with the same directory structure and files as an existing repository.

1. Go to the Honeycomb repository
2. Click on `Use this template` and select `Create a new repository`.



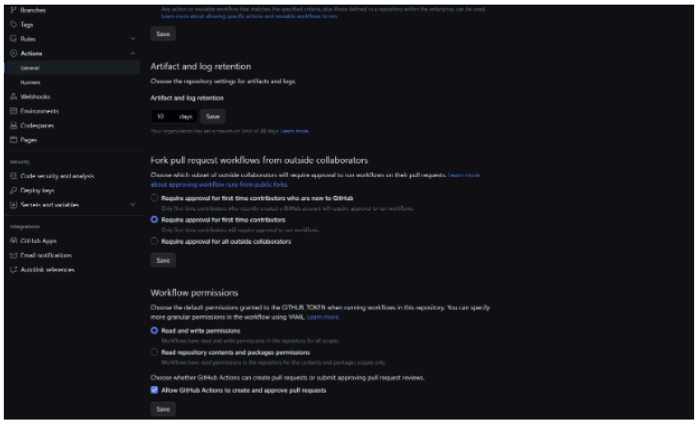
3. Enter the owner, name, and description of your repository and click on `Create repository from template`.



> **NOTE**
> We recommend creating a public repository and leaving `Include all branches` unchecked

4. Ensure the repository's workflow permissions are set to "Read and write permissions"

`Settings -> Actions -> General -> Workflow permissions`

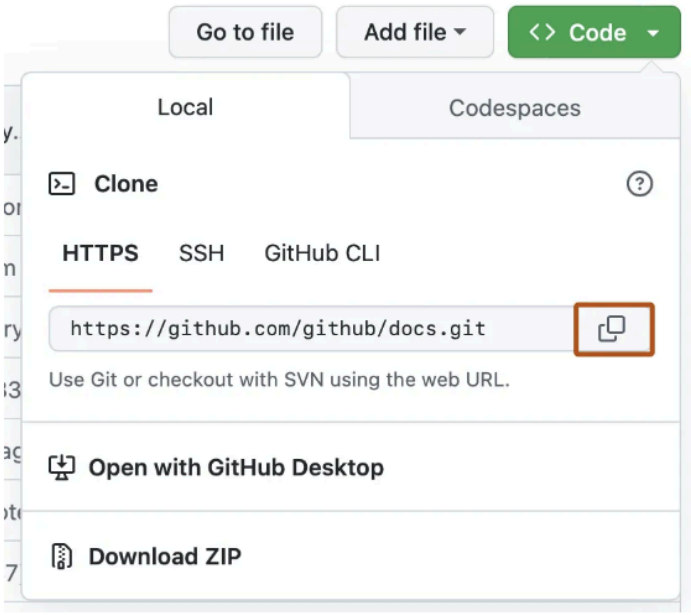


Additional details about template repositories can be found on the Github Docs.

Cloning the Repository

With the repository now setup it can be cloned onto your computer.

1. Navigate to the repository on GitHub.
2. Click the `Code` button and copy the URL



3. Open a terminal and navigate to where you want the cloned directory

Windows macOS

Terminal.app

```
cd 'path/to/directory'
```

4. Clone the repo with the following command

Paste the URL you copied earlier

```
git clone https://github.com/<YOUR-USERNAME>/<YOUR-REPOSITORY>
```

5. Navigate into the cloned repository

The folder is the name of your repository

```
cd <YOUR-REPOSITORY>
```

> NOTE
> Git can be downloaded here if it is not already on your system.

Additional details and alternative methods for cloning a repository can be found on the Github Docs.

Installing Prerequisites

All of the needed programs for Honeycomb must be installed before we can develop our task. We will use a package manager to automatically install them.

See Prerequisites for more information about these programs.

Initial Install

Windows macOS

The most commonly used package manager on macOS is Homebrew.

1. Paste the following command in a macOS Terminal and follow the prompts to install Homebrew.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Windows macOS

The most commonly used package manager on macOS is Homebrew.

1. Paste the following command in a macOS Terminal and follow the prompts to install Homebrew.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Paste the following command and follow the prompts to install the listed programs:

```
brew bundle
```

3. Install Xcode (not available on Homebrew)

```
xcode-select --install
```

> NOTE
> If you are running into issues after installing the packages, please restart your terminal and/or reboot your computer. This should resolve most issues.

Setting Up Node

NVM (Node Version Manager) is a tool for installing and using multiple versions on Node on your computer. It must first be installed:

1. Install NVM

Windows macOS

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.4/install.sh | bash
```

2. Copy the version number listed in `.nvmrc`.
3. Install that version.

```
nvm install <VERSION>
```

4. Use the needed version.

```
nvm use <VERSION>
```

5. Set the current version as your default

```
nvm alias default node
```

> NOTE
> You can skip this step if you've already set a default node version in a different project.

Install Dependencies

There are many Node packages used by Honeycomb that also need to be installed. Node comes with its own package manager to install, update, and maintain these dependencies throughout the development lifecycle.

Install Honeycomb's dependencies

```
npm install
```

Certain Node dependencies are best installed globally. These tools will be available from the command line anywhere on your system.

Install Honeycomb's global dependencies

```
npm install -g electron firebase-tools dotenv-cli lint-staged
```

Run the Task

Run the task in development mode

```
npm run dev
```

Run the Task

Run the task in development mode

```
npm run dev
```

Running the task in development mode enables "hot reloading": changes to the code will immediately be reflected in the app without needing to restart the server.

> **TIP**
> The dev script runs Honeycomb on Electron without any environment variables. Check out the NPM Scripts page for more information on the available development environments.

> **NOTE**
> Honeycomb ships with a modified version of the "simple reaction time task" from the jsPsych tutorial. In the next section we'll create a new task and tell Honeycomb to run it!

Edit the Task

Now that the project is up and running we can make our first changes to the code!

> **INFO**
> The quick start guide details a command line workflow for version control. If you are unfamiliar with Git, we recommend reading the linked page before proceeding.

1) Edit the Project Metadata

1. Create a new branch in a separate terminal

   Create the branch edit-package-json

   ```
   git checkout -b edit-package-json
   ```

2. Open `package.json` and edit it to reflect your app:

   i. `name` is your task's name, generally this is the name of our repository
   ii. `description` should be rewritten to better match your task
   iii. `author` is your lab (or PIs) name, email, and website
   iv. `honeycombVersion` is the number currently in the `version` field
   v. `version` should then be reset to 1.0.0
   vi. `repository` is the link the GitHub repository you created earlier.

   package.json

   ```
   {
     "name": "my-task",
     "description": "A custom task for the Honeycomb platform",
     "author": {
       "name": "My Lab",
       "email": "example@domain.com",
       "url": "https://lab-web-page.com"
     },
     "honeycombVersion": "3.3.0", // Match what was in version!
     "version": "3.3.0",
     "version": "1.0.0",
     "repository": "https://github.com/my-username/my-repository"
   },
   ```

3. Save your changes and commit them to git:

   Commit all changed files with a custom message

   ```
   git commit -a -m "edit package.json with my task's information"
   ```

4. Create and merge a pull request to merge your changes into the `main` branch. Make sure the builds complete successfully before merging!

2) Add a file for the task

1. Bring your branch up to date with the `main` branch

   Switch to the main branch

   ```
   git checkout main
   ```

   Bring changes from GitHub into your local repository

   ```
   git pull
   ```

2. Create a new branch (replace `task-name` with the name of your task)

Check out a new branch

```
git checkout -b add-<task-name>-file
```

3. Add a new file inside `src/experiment/` with the same name as your task

4. Save your changes and commit them to git:

Add the new file to Git

```
git add .
```

Commit all changed files with a custom message

```
git commit -a -m "feat: adds file for the <task name> task"
```

5. Add a `taskNameOptions` object to the new file (replace `taskName` with the name of your task)

taskName.js

```
/**
 * Experiment-wide settings for jsPsych: https://www.jspsych.org/7.3/overview/experiment-options/
 * Note that Honeycomb combines these with other options required for Honeycomb to operate correctly
 */
export const taskNameOptions = {
  // Called when every trial finishes
  on_trial_finish: function (data) {
    console.log(`Trial ${data.internal_node_id} just finished:`, data);
  },
  // Called when the experiment finishes
  on_finish: function (data) {
    console.log("The experiment has finished:", data);
    // Reload the page for another run-through of the experiment
    window.location.reload();
  },
};
```

6. Add a `buildTaskNameFunction` to the new file (replace `TaskName` with the name of your task)

taskName.js

```
/**
 * This timeline builds the example reaction time task from the jsPsych tutorial.
 * Take a look at how the code here compares to the jsPsych documentation!
 * See the jsPsych documentation for more: https://www.jspsych.org/7.3/tutorials/rt-task/
 *
 * @param {Object} jsPsych The jsPsych instance being used to run the task
 * @returns {Object} A jsPsych timeline object
 */
export function buildTaskNameTimeline(jsPsych) {}
```

7. Save your changes and commit them to git:

Commit all changed files with a custom message

```
git commit -a -m "feat: adds taskNameOptions and buildTaskNameTimeline to taskName.js"
```

8. Edit `src/experiment/index.js` to use the new file

src/experiment/index.js

```
import { buildHoneycombTimeline, honeycombOptions } from "./honeycomb";
import { buildTaskNameTimeline, taskNameOptions } from "./taskName";

// ...

export const jsPsychOptions = honeycombOptions;
export const jsPsychOptions = taskNameOptions;

// ...

export function buildTimeline(jsPsych, studyID, participantID) {
  console.log(
    `Building timeline for participant ${participantID} on study ${studyID}`
  );

  const timeline = buildHoneycombTimeline(jsPsych);
  const timeline = buildTaskNameTimeline(jsPsych);
  return timeline;
}
```

9. Run the format script to make sure the code is formatted correctly

```
npm run format
```

10. Save your changes and commit them to git:

10. Save your changes and commit them to git:

Commit all changed files with a custom message

```
git commit -a -m "fix: Use new task's file"
```

11. Create and merge a pull request to merge your changes into the `main` branch. Make sure the builds complete successfully before merging!

3) Add some trials to the task

1. Bring your branch up to date with the `main` branch

Switch to the main branch

```
git checkout main
```

Bring changes from GitHub into your local repository

```
git pull
```

2. Create a new branch

Bring changes from GitHub into your local repository

```
git checkout -b add-start-procedure
```

3. Add the start procedure to the `buildTaskNameTimeline` function in the file you created earlier

taskName.js

```
import { buildStartProcedure } from "./procedures/startProcedure";

// ...

export function buildTaskNameTimeline(jsPsych) {
  // Build the trials that make up the start procedure
  const startProcedure = buildStartProcedure(jsPsych);

  const timeline = [startProcedure];
  return timeline;
}

// ...
```

4. Save your changes and commit them to git:

Commit all changed files with a custom message

```
git commit -a -m "feat: adds startProcedure to the task"
```

5. Edit the text for the task's name

src/config/language.json

```
{
  "name": "taskName"
  // ...
}
```

> **TIP**
>
> The text for the introduction trial is in `src/config/language.json` under the `trials` and `introduction` key.
>
> src/config/language.json
>
> ```
> {
>   "name": "taskName"
>   // ...
>   "trials": {
>     "introduction": "Welcome to the experiment. Press any key to begin."
>     // ...
>   },
>   // ...
> }
> ```

6. Save your changes and commit them to git:

Commit all changed files with a custom message

```
git commit -a -m "feat: Updates the language for the startProcedure of the task"
```

Next Step

7. Add the end procedure to the `buildTaskNameTimeline` function in the file you created earlier

taskName.js

```
import { buildStartProcedure } from "./procedures/startProcedure";
import { buildEndProcedure } from "./procedures/endProcedure";

// ...

export function buildTaskNameTimeline(jsPsych) {
  // Build the trials that make up the start procedure
  const startProcedure = buildStartProcedure(jsPsych);

  // Builds the trials that make up the end procedure
  const endProcedure = buildEndProcedure(jsPsych);

  const timeline = [startProcedure, endProcedure];
  return timeline;
}
// ...
```

TIP

The text for the conclusion trial is in `src/config/language.json` under the `trials` and `conclusion` key.

src/config/language.json

```
{
  "name": "taskName"
  // ...
  "trials": {
    // ...
    "conclusion": "Welcome to the experiment. Press any key to begin."
  },
}
```

8. Run a format to make sure the code is formatted correctly

```
npm run format
```

9. Save your changes and commit them to git:

Commit all changed files with a custom message

```
git commit -a -m "feat: adds endProcedure to the task"
```

10. Create and merge a pull request to merge your changes into the `main` branch. Make sure the builds complete successfully before merging!

Next Steps

- The Firebase page explains how to set up your task with Firebase.
- The Environment Variables page explains how to configure your task for deployment to multiple scenarios.
- The NPM Scripts page lists every script you can run and which environment they use.