Use the terminal or an Anaconda Prompt for the following steps:

1. To create an environment:

```
2. conda create --name myenv
```

Note

Replace `myenv` with the environment name.

3. When conda asks you to proceed, type `y`:

```
4. proceed ([y]/n)?
```

This creates the myenv environment in `/envs/`. No packages will be installed in this environment.

1. To create an environment with a specific version of Python:

```
2. conda create -n myenv python=3.6
```

3. To create an environment with a specific package:

```
4. conda create -n myenv scipy
```

OR:

```
conda create -n myenv python
conda install -n myenv scipy
```

5. To create an environment with a specific version of a package:

```
6. conda create -n myenv scipy=0.15.0
```

OR:

```
conda create -n myenv python
conda install -n myenv scipy=0.15.0
```

7. To create an environment with a specific version of Python and multiple packages:

```
conda create -n myenv python=3.6 scipy=0.15.0 astroid babel
```

Tip

Install all the programs that you want in this environment at the same time. Installing 1 program at a time can lead to dependency conflicts.

To automatically install pip or another program every time a new environment is created, add the default programs to the create_default_packages section of your `.condarc` configuration file. The default packages are installed every time you create a new environment. If you do not want the default packages installed in a particular environment, use the `--no-default-packages` flag:

```
conda create --no-default-packages -n myenv python
```

Tip

You can add much more to the `conda create` command. For details, run `conda create --help`.

Creating an environment from an environment.yml file

Use the terminal or an Anaconda Prompt for the following steps:

1. Create the environment from the `environment.yml` file:
2. `conda env create -f environment.yml`

   The first line of the `yml` file sets the new environment's name. For details see Creating an environment file manually.
3. Activate the new environment: `conda activate myenv`
4. Verify that the new environment was installed correctly:
5. `conda env list`

You can also use `conda info --envs`.

Specifying a location for an environment

You can control where a conda environment lives by providing a path to a target directory when creating the environment. For example, the following command will create a new environment in a subdirectory of the current working directory called `envs`:

```
conda create --prefix ./envs jupyterlab=3.2 matplotlib=3.5 numpy=1.21
```

You then activate an environment created with a prefix using the same command used to activate environments created by name:

```
conda activate ./envs
```

Specifying a path to a subdirectory of your project directory when creating an environment has the following benefits:

- It makes it easy to tell if your project uses an isolated environment by including the environment as a subdirectory.
- It makes your project more self-contained as everything, including the required software, is contained in a single project directory.

An additional benefit of creating your project's environment inside a subdirectory is that you can then use the same name for all your environments. If you keep all of your environments in your `envs` folder, you'll have to give each environment a different name.

There are a few things to be aware of when placing conda environments outside of the default `envs` folder.

1. Conda can no longer find your environment with the `--name` flag. You'll generally need to pass the `--prefix` flag along with the environment's full path to find the environment.
2. Specifying an install path when creating your conda environments makes it so that your command prompt is now prefixed with the active environment's absolute path rather than the environment's name.

After activating an environment using its prefix, your prompt will look similar to the following:

```
(/absolute/path/to/envs) $
```

This can result in long prefixes:

```
(/Users/USER_NAME/research/data-science/PROJECT_NAME/envs) $
```

To remove this long prefix in your shell prompt, modify the env_prompt setting in your `.condarc` file:

```
$ conda config --set env_prompt '({name})'
```

This will edit your `.condarc` file if you already have one or create a `.condarc` file if you do not.

Now your command prompt will display the active environment's generic name, which is the name of the environment's root folder:

```
$ cd project-directory
$ conda activate ./env
(env) project-directory $
```

Updating an environment

You may need to update your environment for a variety of reasons. For example, it may be the case that:

- one of your core dependencies just released a new version (dependency version number update).
- you need an additional package for data analysis (add a new dependency).
- you have found a better package and no longer need the older package (add new dependency and remove old dependency).

If any of these occur, all you need to do is update the contents of your `environment.yml` file accordingly and then run the following command:

```
$ conda env update --prefix ./env --file environment.yml  --prune
```

Note

The `--prune` option causes conda to remove any dependencies that are no longer required from the environment.

Cloning an environment

Use the terminal or an Anaconda Prompt for the following steps:

You can make an exact copy of an environment by creating a clone of it:

```
conda create --name myclone --clone myenv
```

Note

Replace `myclone` with the name of the new environment. Replace `myenv` with the name of the existing environment that you want to copy.

To verify that the copy was made:

```
conda info --envs
```

In the environments list that displays, you should see both the source environment and the new copy.

Building identical conda environments

You can use explicit specification files to build an identical conda environment on the same operating system platform, either on the same machine or on a different machine.

Use the terminal or an Anaconda Prompt for the following steps:

1. Run `conda list --explicit` to produce a spec list such as:

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: osx-64
@EXPLICIT
https://repo.anaconda.com/pkgs/free/osx-64/mkl-11.3.3-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/numpy-1.11.1-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/openssl-1.0.2h-1.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/pip-8.1.2-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/python-3.5.2-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/readline-6.2-2.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/setuptools-25.1.6-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/sqlite-3.13.0-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/tk-8.5.18-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/wheel-0.29.0-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/xz-5.2.2-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/zlib-1.2.8-3.tar.bz2
```

18. To create this spec list as a file in the current working directory, run:

```
conda list --explicit > spec-file.txt
```

Note

You can use `spec-file.txt` as the filename or replace it with a filename of your choice.

An explicit spec file is not usually cross platform, and therefore has a comment at the top such as `# platform: osx-64` showing the platform where it was created. This platform is the one where this spec file is known to work. On other platforms, the packages specified might not be available or dependencies might be missing for some of the key packages already in the spec.

To use the spec file to create an identical environment on the same machine or another machine:

```
conda create --name myenv --file spec-file.txt
```

To use the spec file to install its listed packages into an existing environment:

```
conda install --name myenv --file spec-file.txt
```

Conda does not check architecture or dependencies when installing from a spec file. To ensure that the packages work correctly, make sure that the file was created from a working environment, and use it on the same architecture, operating system, and platform, such as linux-64 or osx-64.

Activating an environment

Activating environments is essential to making the software in the environments work well. Activation entails two primary functions: adding entries to PATH for the environment and running any activation scripts that the environment may contain. These activation scripts are how packages can set arbitrary environment variables that may be necessary for their operation. You can also use the config API to set environment variables.

When installing Anaconda, you have the option to “Add Anaconda to my PATH environment variable.” This is not recommended because the add to PATH option appends Anaconda to PATH. When the installer appends to PATH, it does not call the activation scripts.

On Windows, PATH is composed of two parts, the system PATH and the user PATH. The system PATH always comes first. When you install Anaconda for Just Me, we add it to the user PATH. When you install for All Users, we add it to the system PATH. In the former case, you can end up with system PATH values taking precedence over our entries. In the latter case, you do not. We do not recommend multi-user installs.

Activation prepends to PATH. This only takes effect when you have the environment active so it is local to a terminal session, not global.

To activate an environment: `conda activate myenv`

Note

Replace `myenv` with the environment name or directory path.

Conda prepends the path name `myenv` onto your system command.

You may receive a warning message if you have not activated your environment:

```
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation.
```

If you receive this warning, you need to activate your environment. To do so on Windows, run: `c:\Anaconda3\Scripts\activate base` in Anaconda Prompt.

Windows is extremely sensitive to proper activation. This is because the Windows library loader does not support the concept of libraries and executables that know where to search for their dependencies (RPATH). Instead, Windows relies on a dynamic-link library search order.

If environments are not active, libraries won't be found and there will be lots of errors. HTTP or SSL errors are common errors when the Python in a child environment can't find the necessary OpenSSL library.

Conda itself includes some special workarounds to add its necessary PATH entries. This makes it so that it can be called without activation or with any child environment active. In general, calling any executable in an environment without first activating that environment will likely not work. For the ability to run executables in activated environments, you may be interested in the `conda run` command.

If you experience errors with PATH, review our troubleshooting.

Conda init

Earlier versions of conda introduced scripts to make activation behavior uniform across operating systems. Conda 4.4 allowed `conda activate myenv`. Conda 4.6 added extensive initialization support so that conda works faster and less disruptively on a wide variety of shells (bash, zsh, csh, fish, xonsh, and more). Now these shells can use the `conda activate` command.

Removing the need to modify PATH makes conda less disruptive to other software on your system. For more information, read the output from `conda init --help`.

One setting may be useful to you when using `conda init` is:

```
auto_activate_base: bool
```

This setting controls whether or not conda activates your base environment when it first starts up. You'll have the `conda` command available either way, but without activating the environment, none of the other programs in the environment will be available until the environment is activated with `conda activate base`. People sometimes choose this setting to speed up the time their shell takes to start up or to keep conda-installed software from automatically hiding their other software.

Nested activation

By default, `conda activate` will deactivate the current environment before activating the new environment and reactivate it when deactivating the new environment. Sometimes you may want to leave the current environment PATH entries in place so that you can continue to easily access command-line programs from the first environment. This is most commonly encountered when common command-line utilities are installed in the base environment. To retain the current environment in the PATH, you can activate the new environment using:

```
conda activate --stack myenv
```

If you wish to always stack when going from the outermost environment, which is typically the base environment, you can set the `auto_stack` configuration option:

```
conda config --set auto_stack 1
```

You may specify a larger number for a deeper level of automatic stacking, but this is not recommended since deeper levels of stacking are more likely to lead to confusion.

Environment variable for DLL loading verification

If you don't want to activate your environment and you want Python to work for DLL loading verification, then follow the troubleshooting directions.

Warning

If you choose not to activate your environment, then loading and setting environment variables to activate scripts will not happen. We only support activation.

Deactivating an environment

To deactivate an environment, type: `conda deactivate`

Conda removes the path name for the currently active environment from your system command.

Note

To simply return to the base environment, it's better to call `conda activate` with no environment specified, rather than to try to deactivate. If you run `conda deactivate` from your base environment, you may lose the ability to run conda at all. Don't worry, that's local to this shell - you can start a new one. However, if the environment was activated using `--stack` (or was automatically stacked) then it is better to use `conda deactivate`.

Determining your current environment

Use the terminal or an Anaconda Prompt for the following steps.

By default, the active environment---the one you are currently using---is shown in parentheses () or brackets [] at the beginning of your command prompt:

```
(myenv) $
```

If you do not see this, run:

```
conda info --envs
```

In the environments list that displays, your current environment is highlighted with an asterisk (*).

By default, the command prompt is set to show the name of the active environment. To disable this option:

```
conda config --set changeps1 false
```

To re-enable this option:

```
conda config --set changeps1 true
```

Viewing a list of your environments

To see a list of all of your environments, in your terminal window or an Anaconda Prompt, run:

```
conda info --envs
```

OR

```
conda env list
```

A list similar to the following is displayed:

```
conda environments:
myenv                  /home/username/miniconda/envs/myenv
snowflakes             /home/username/miniconda/envs/snowflakes
bunnies                /home/username/miniconda/envs/bunnies
```

If this command is run by an administrator, a list of all environments belonging to all users will be displayed.

Viewing a list of the packages in an environment

To see a list of all packages installed in a specific environment:

- If the environment is not activated, in your terminal window or an Anaconda Prompt, run:

```
conda list -n myenv
```

- If the environment is activated, in your terminal window or an Anaconda Prompt, run:

```
conda list
```

- To see if a specific package is installed in an environment, in your terminal window or an Anaconda Prompt, run:

```
conda list -n myenv scipy
```

Using pip in an environment

To use pip in your environment, in your terminal window or an Anaconda Prompt, run:

```
conda install -n myenv pip
conda activate myenv
pip <pip_subcommand>
```

Issues may arise when using pip and conda together. When combining conda and pip, it is best to use an isolated conda environment. Only after conda has been used to install as many packages as possible should pip be used to install any remaining software. If modifications are needed to the environment, it is best to create a new environment rather than running conda after pip. When appropriate, conda and pip requirements should be stored in text files.

We recommend that you:

Use pip only after conda

- Install as many requirements as possible with conda then use pip.
- Pip should be run with `--upgrade-strategy only-if-needed` (the default).
- Do not use pip with the `--user` argument, avoid all users installs.

Use conda environments for isolation

- Create a conda environment to isolate any changes pip makes.
- Environments take up little space thanks to hard links.
- Care should be taken to avoid running pip in the root environment.

Recreate the environment if changes are needed

- Once pip has been used, conda will be unaware of the changes.
- To install additional conda packages, it is best to recreate the environment.

Store conda and pip requirements in text files

- Package requirements can be passed to conda via the `--file` argument.
- Pip accepts a list of Python packages with `-r` or `--requirements`.
- Conda env will export or create environments based on a file with conda and pip requirements.

Setting environment variables

If you want to associate environment variables with an environment, you can use the config API. This is recommended as an alternative to using activate and deactivate scripts since those are an execution of arbitrary code that may not be safe.

First, create your environment and activate it:

```
conda create -n test-env
conda activate test-env
```

To list any variables you may have, run `conda env config vars list`.

To set environment variables, run `conda env config vars set my_var=value`.

Once you have set an environment variable, you have to reactivate your environment: `conda activate test-env`.

To check if the environment variable has been set, run `echo $my_var` (`echo %my_var%` on Windows) or `conda env config vars list`.

When you deactivate your environment, you can use those same commands to see that the environment variable goes away.

You can specify the environment you want to affect using the `-n` and `-p` flags. The `-n` flag allows you to name the environment and `-p` allows you to specify the path to the environment.

To unset the environment variable, run `conda env config vars unset my_var -n test-env`.

When you deactivate your environment, you can see that environment variable goes away by rerunning `echo my_var` or `conda env config vars list` to show that the variable name is no longer present.

Environment variables set using `conda env config vars` will be retained in the output of `conda env export`. Further, you can declare environment variables in the environment.yml file as shown here:

```
name: env-name
```

```
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.7
  - codecov
variables:
  VAR1: valueA
  VAR2: valueB
```

Saving environment variablesℑ

Conda environments can include saved environment variables.

Suppose you want an environment "analytics" to store both a secret key needed to log in to a server and a path to a configuration file. The sections below explain how to write a script named `env_vars` to do this on Windows and macOS or Linux.

This type of script file can be part of a conda package, in which case these environment variables become active when an environment containing that package is activated.

You can name these scripts anything you like. However, multiple packages may create script files, so be sure to use descriptive names that are not used by other packages. One popular option is to give the script a name in the form `packagename-scriptname.sh`, or on Windows, `packagename-scriptname.bat`.

Windowsℑ

1. Locate the directory for the conda environment in your Anaconda Prompt by running in the command shell `%CONDA_PREFIX%`.
2. Enter that directory and create these subdirectories and files:

```
3.  cd %CONDA_PREFIX%
4.  mkdir .\etc\conda\activate.d
5.  mkdir .\etc\conda\deactivate.d
6.  type NUL > .\etc\conda\activate.d\env_vars.bat
7.  type NUL > .\etc\conda\deactivate.d\env_vars.bat
```

8. Edit `.\etc\conda\activate.d\env_vars.bat` as follows:

```
9.  set MY_KEY='secret-key-value'
10. set MY_FILE=C:\path\to\my\file
```

11. Edit `.\etc\conda\deactivate.d\env_vars.bat` as follows:

```
set MY_KEY=
set MY_FILE=
```

When you run `conda activate analytics`, the environment variables `MY_KEY` and `MY_FILE` are set to the values you wrote into the file. When you run `conda deactivate`, those variables are erased.

macOS and Linuxℑ

1. Locate the directory for the conda environment in your terminal window by running in the terminal `echo $CONDA_PREFIX`.
2. Enter that directory and create these subdirectories and files:

```
cd $CONDA_PREFIX
mkdir -p ./etc/conda/activate.d
mkdir -p ./etc/conda/deactivate.d
touch ./etc/conda/activate.d/env_vars.sh
touch ./etc/conda/deactivate.d/env_vars.sh
```

8. Edit `./etc/conda/activate.d/env_vars.sh` as follows:

```
#!/bin/sh

export MY_KEY='secret-key-value'
export MY_FILE=/path/to/my/file/
```

13. Edit `./etc/conda/deactivate.d/env_vars.sh` as follows:

```
#!/bin/sh

unset MY_KEY
unset MY_FILE
```

When you run `conda activate analytics`, the environment variables `MY_KEY` and `MY_FILE` are set to the values you wrote into the file. When you run `conda deactivate`, those variables are erased.

Sharing an environmentℑ

You may want to share your environment with someone else---for example, so they can re-create a test that you have done. To allow them to quickly reproduce your environment, with all of its packages and versions, give them a copy of your `environment.yml` file.

Exporting the environment.yml fileℑ

Note

If you already have an `environment.yml` file in your current directory, it will be overwritten during this task.

1. Activate the environment to export: `conda activate myenv`

   Note

   Replace `myenv` with the name of the environment.

2. Export your active environment to a new file:

```
3. conda env export > environment.yml
```

   Note

   This file handles both the environment's pip packages and conda packages.

4. Email or copy the exported `environment.yml` file to the other person.

Exporting an environment file across platforms

If you want to make your environment file work across platforms, you can use the `conda env export --from-history` flag. This will only include packages that you've explicitly asked for, as opposed to including every package in your environment.

For example, if you create an environment and install Python and a package:

```
conda install python=3.7 codecov
```

This will download and install numerous additional packages to solve for dependencies. This will introduce packages that may not be compatible across platforms.

If you use `conda env export`, it will export all of those packages. However, if you use `conda env export --from-history`, it will only export those you specifically chose:

```
(env-name) ➜  ~ conda env export --from-history
name: env-name
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.7
  - codecov
prefix: /Users/username/anaconda3/envs/env-name
```

Note

If you installed Anaconda 2019.10 on macOS, your prefix may be `/Users/username/opt/envs/env-name`.

Creating an environment file manually

You can create an environment file (`environment.yml`) manually to share with others.

EXAMPLE: A simple environment file:

```
name: stats
dependencies:
  - numpy
  - pandas
```

EXAMPLE: A more complex environment file:

```
name: stats2
channels:
  - javascript
dependencies:
  - python=3.6   # or 2.7
  - bokeh=0.9.2
  - numpy=1.9.*
  - nodejs=0.10.*
  - flask
  - pip:
    - Flask-Testing
```

Note

Note the use of the wildcard * when defining the patch version number. Defining the version number by fixing the major and minor version numbers while allowing the patch version number to vary allows us to use our environment file to update our environment to get any bug fixes whilst still maintaining consistency of software environment.

You can exclude the default channels by adding `nodefaults` to the channels list.

```
channels:
  - javascript
  - nodefaults
```

This is equivalent to passing the `--override-channels` option to most `conda` commands.

Adding `nodefaults` to the channels list in `environment.yml` is similar to removing `defaults` from the channels list in the `.condarc` file. However, changing `environment.yml` affects only one of your conda environments while changing `.condarc` affects them all.

For details on creating an environment from this `environment.yml` file, see Creating an environment from an environment.yml file.

Restoring an environment

Conda keeps a history of all the changes made to your environment, so you can easily "roll back" to a previous version. To list the history of each change to the current environment: `conda list --revisions`

To restore environment to a previous revision: `conda install --revision=REVNUM` or `conda install --rev REVNUM`.

Note

Replace REVNUM with the revision number.

Example: If you want to restore your environment to revision 8, run `conda install --rev 8`.

Removing an environment

To remove an environment, in your terminal window or an Anaconda Prompt, run:

```
conda remove --name myenv --all
```

You may instead use `conda env remove --name myenv`.

To verify that the environment was removed, in your terminal window or an Anaconda Prompt, run:

```
conda info --envs
```

The environments list that displays should not show the removed environment.