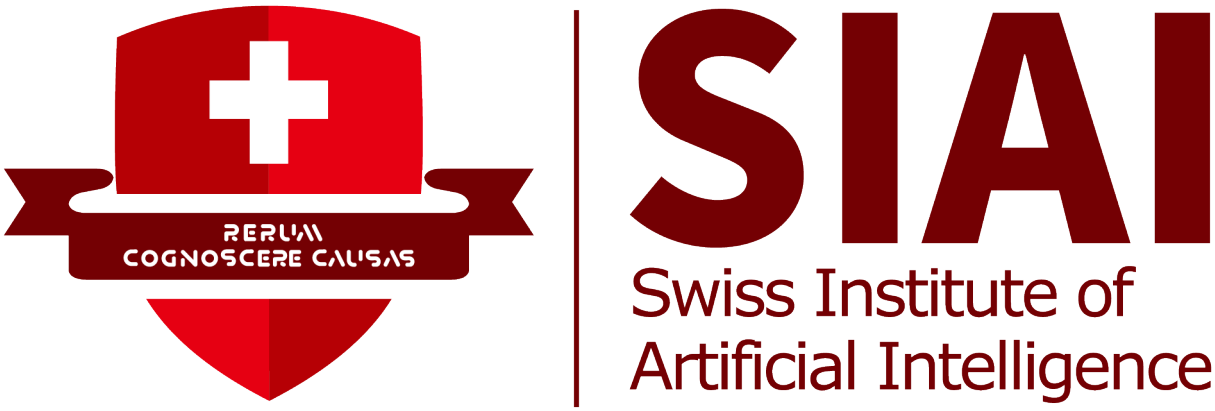
Issues in Computer Programming

Python Programming for Data Science

Swiss Institute of Artificial Intelligence (SIAI)

August 18, 2021



Contents

Chapter 1

Programming Python on Windows

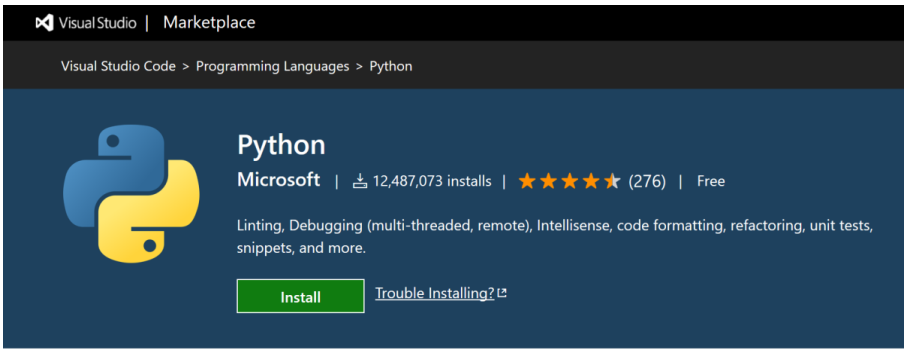
1.1 For Single Python Project with Yourself

1.1.0 Prerequisites

Preinstalled Software

Following software is necessary to be installed in advance.

- Python 3.9.x[1] from python.org
- Visual Studio Code (VS Code) from https://code.visualstudio.com/Download (**System Installer 64bit**)
- Python extension for Visual Studio Code from Extensions for the Visual Studio family of products[2]



- Jupyter notebook package

VS Code Configurations

If your VS code display language is not English, please change the display language to English.

```
The Command Palette (Ctrl+Shift+P or F1)
> configure display language
> Select Display Language > en
```

- Changing the Display Language

[1] **NOTICE: Why Python 3.9? The current latest stable version**

[2] Microsoft. *Python extension for Visual Studio Code*. URL: https://marketplace.visualstudio.com/items?itemName=ms-python.python.

Press `Ctrl+Shift+P` to bring up the `Command Palette` then start typing ”display” to filter and display the `Configure Display Language` command.

Press `Enter` and a list of installed languages by `locale` is displayed, with the current `locale` highlighted.

1.1.1 Running Python Scripts with Hello World Example in VS Code

(1) Making a folder such as: `%USERPROFILE%\workspaces\hello`[3]

(2) Opening the folder with VS Code

(3) **Selecting a Python Interpreter: pre-installed Python 3.9.x**

(4) **Creating and editing a Python Script file**: `hello.py`

(5) **Running Python Scripts in VS Code**

(3) Selecting a Python Interpreter: pre-installed Python 3.9.x

Select a Python interpreter to run Python scripts in VS Code if an interpreter is not selected.

```
The Command Palette (Ctrl+Shift+P or F1)
> python: select interpreter
> Python 3.9.x 64-bit
```

- Select a Python interpreter

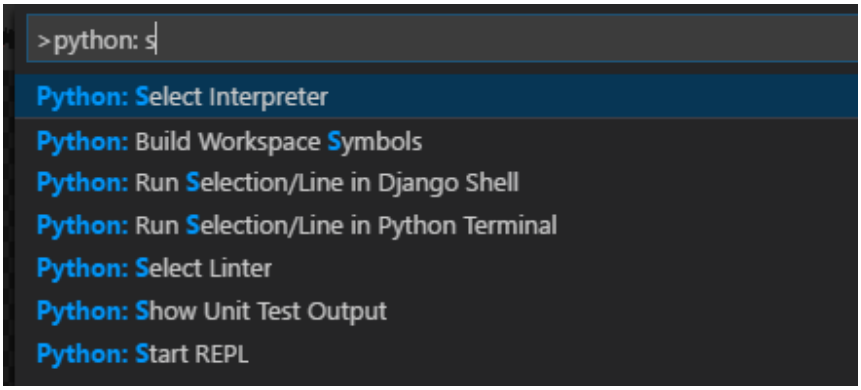
  From within VS Code, select a Python 3 interpreter by opening the `Command Palette (Ctrl+Shift+P)`, start typing the `Python: Select Interpreter` command to search, then select the command. You can also use the `Select Python Environment` option on the Status Bar if available (it may already show a selected interpreter, too):

- Select and activate an environment

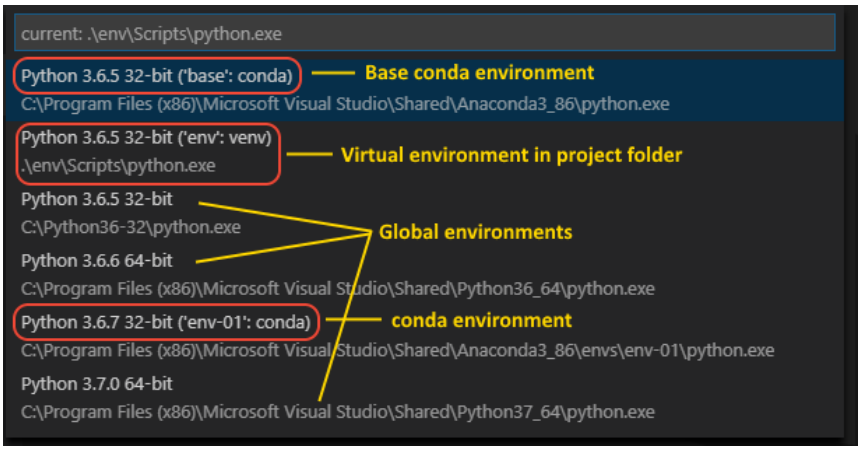
  By default, the Python extension looks for and uses the first Python interpreter it finds in the system path. To select a specific environment, use the `Python: Select Interpreter` command

[3] It is highly recommended to create `workspaces` folder to manage programming projects.

from the `Command Palette (Ctrl+Shift+P)`.



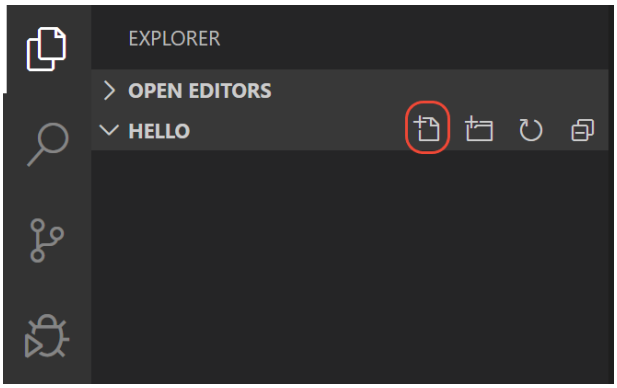
The `Python: Select Interpreter` command displays a list of available global environments, conda environments, and virtual environments. (See the Where the extension looks for environments section for details, including the distinctions between these types of environments.) The following image, for example, shows several Anaconda and CPython installations along with a conda environment and a virtual environment (`env`) that's located within the workspace folder:



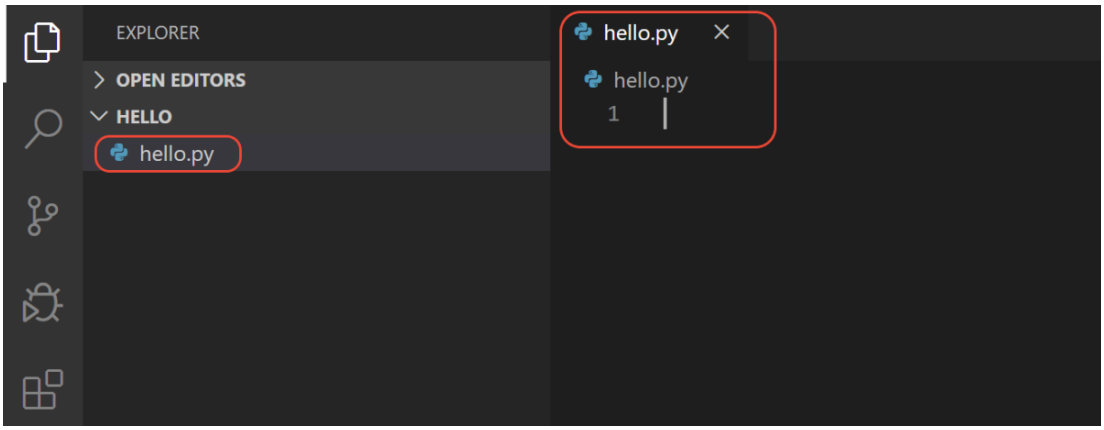
(4) Creating and editing a Python Script file: hello.py

- Create a Python Hello World source code file

From the File Explorer toolbar, select the `New File` button on the `hello` folder:



Name the file `hello.py`, and it automatically opens in the editor:



Now that you have a code file in your Workspace, enter the following source code in `hello.py`:

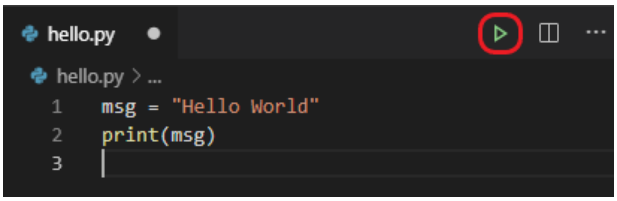
```
msg = "Hello World"
print(msg)
```

(5) Running Python Scripts in VS Code

Run Hello World

- Run Python File in **`Terminal`**

  It's simple to run hello.py with Python. Just click the `Run Python File in Terminal` play button in the top-right side of the editor.



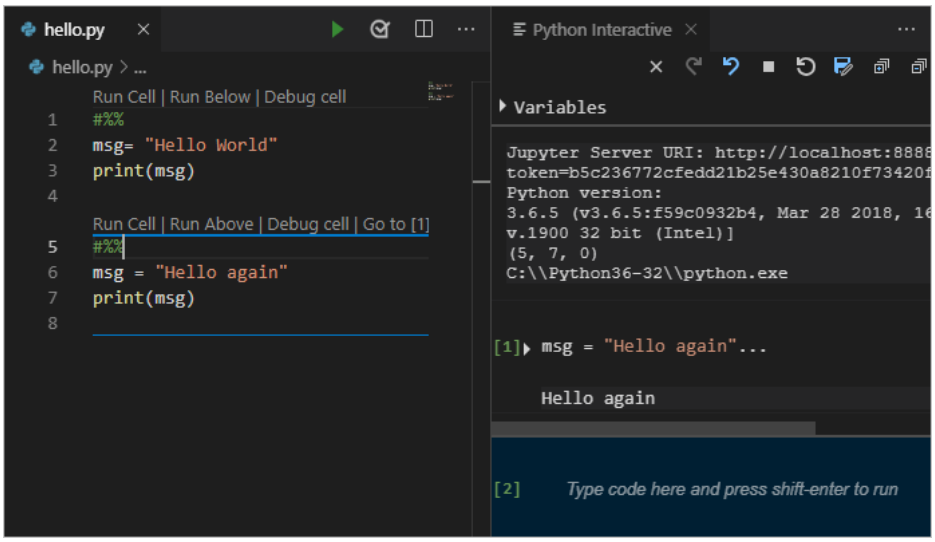
- Run Selection/Line in **`Python Terminal (Shift + Enter)`** (Python Shell; REPL)

  Select one or more lines, then press `Shift+Enter` or right-click and select `Run Selection/Line in Python Terminal`. This command is convenient for testing just a part of a file.

- Run Selection/Line in **`Interactive Window (Shift + Enter)`** (Jupyter notebook code cells)

  1) Adding `# %%` comment in Python source code to use code cells
     Python Interactive window, Jupyter Notebooks in VS Code

     You define Jupyter-like code cells within Python code using a `# %%` comment:

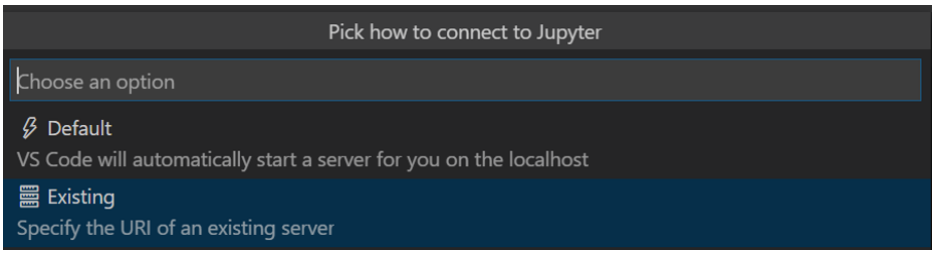


2) Running and Connecting a local Jupyter server

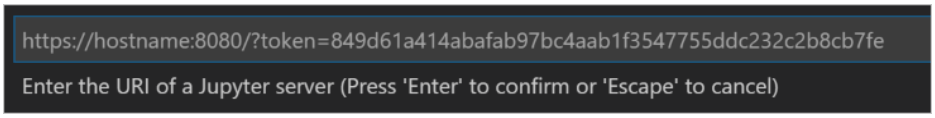
```
The Command Palette (Ctrl+Shift+P or F1)
> Jupyter: Specify local or remote Jupyter server for connections
> Default - VS Code will automatically start a server for you on the localhost
```

Python Interactive window: Connect to a remote Jupyter server,
Jupyter Notebooks in VS Code: Connect to a remote Jupyter server

1. Run the `Jupyter: Specify local or remote Jupyter server for connections` command from the Command Palette (Ctrl+Shift+P).
2. When prompted to `Pick how to connect to Jupyter`, select `Existing: Specify the URI of an existing server`.



3. When prompted to `Enter the URI of a Jupyter server`, provide the server's URI (hostname) with the authentication token included with a `?token=` URL parameter. (If you start the server in the VS Code terminal with an authentication token enabled, the URL with the token typically appears in the terminal output from where you can copy it.) Alternatively, you can specify a username and password after providing the URI.

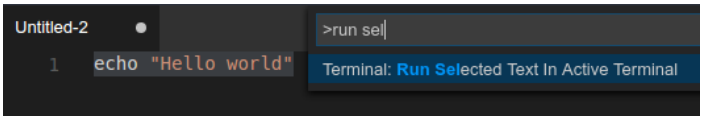


- Run Selection/Line in Python Terminal using **`Run Selected Text`**

1) Executing the `Run Selected Text In Active Terminal` command in VS Code

Run Selected Text

> To use the runSelectedText command, select text in an editor and run the command `Terminal: Run Selected Text in Active Terminal` via the `Command Palette (Ctrl+Shift+P)`:



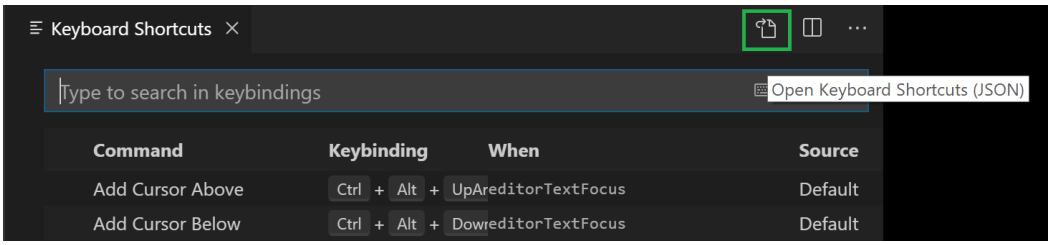
> The terminal will attempt to run the selected text.

2) Adding a keybinding in VS Code for the runSelectedText

```
File > Preferences > Keyboard Shortcuts
(or the Gear icon in the Activity Bar > Keyboard Shortcuts)
> Type to search in keybindings: "workbench.action.terminal.runSelectedText"
> double click "Terminal: Run Selected Text In Active Terminal"
> Press desired key combination and then press ENTER: Ctrl + r
```

3) Adding a keyborad rule in `keybindings.json`
Send text from a keybinding, Advanced customization

> To configure keyboard shortcuts through the JSON file, open `Keyboard Shortcuts` editor and select the `Open Keyboard Shortcuts (JSON)` button on the right of the editor title bar.



Add below code in the `keybindings.json`:

```
[
    {
        "key": "ctrl+r",
        "command": "workbench.action.terminal.runSelectedText",
        "when": "editorTextFocus == true"
    }
]
```

- Run Python Scripts in Command Line using the `Run Selected Text` command:

`1-python-on-windows/python_hello_world.bat`

```
python -c "print('Hello World')"
```

1.1.2 Source Coding for Writing Large Programs

(1) Making `data` folder to save datasets

(2) Making `src` folder to save source code: Multiple Packages (Folders) and Multiple Modules (Files) within src folder

(1) Making `data` folder to save datasets

- Loading Iris Dataset using scikit-learn function:
  `1-python-on-windows/iris_data_1.py`

```
2  from sklearn import datasets
3
4  iris = datasets.load_iris()
5
6  X = iris.data
7  y = iris.target
8
9  print(X.shape)
10 print(y.shape)
```

- Loading Iris Dataset using Pandas function:
  `1-python-on-windows/iris_data_2.py`

```
2  import pandas as pd
3
4  # NOTICE: FileNotFoundError: [Errno 2] No such file or directory: 'data/Iris.csv'
5  #           if the current working directory is in src/
6  df = pd.read_csv("data/Iris.csv")
7
8  print(df.shape)
```

- Loading Iris Dataset using Pandas function with Path variable:
  `1-python-on-windows/iris_data_3.py`

```
2  import os
3  from pathlib import Path
4
5  import pandas as pd
6
7  PROJ_ROOT = None
8
9  # script mode
10 if "__file__" in locals() or "__file__" in globals():
11     PROJ_ROOT = os.path.dirname(Path(__file__).resolve())
12 # interactive mode
13 else:
14     # NOTICE: The current working directory should be the project root folder in
       Python Terminal
15     PROJ_ROOT = os.path.dirname(os.path.realpath("__file__"))
16
17 DATA_ROOT = os.path.join(PROJ_ROOT, "data")
18
19 df = pd.read_csv(f"{DATA_ROOT}/Iris.csv")
20
21 print(df.shape)
```

- Loading Iris Dataset using Pandas function with Path variable from function:
  `1-python-on-windows/iris_data_4.py`

```
import os
from pathlib import Path

import pandas as pd


def get_proj_root():
    return (
        os.path.dirname(Path(__file__).resolve())
        if "__file__" in locals() or "__file__" in globals()
        # NOTICE: The current working directory should be the project root folder in Python Terminal
        else os.path.dirname(os.path.realpath("__file__"))
    )


PROJ_ROOT = get_proj_root()
DATA_ROOT = os.path.join(PROJ_ROOT, "data")

df = pd.read_csv(f"{DATA_ROOT}/Iris.csv")

print(df.shape)
```

- Loading Iris Dataset using Pandas function with Path variable from module:
  `1-python-on-windows/settings_1.py`

```
import os
from pathlib import Path


def get_proj_root():
    return (
        os.path.dirname(Path(__file__).resolve())
        if "__file__" in locals() or "__file__" in globals()
        # NOTICE: The current working directory should be the project root folder in Python Terminal
        else os.path.dirname(os.path.realpath("__file__"))
    )


PROJ_ROOT = get_proj_root()
```

`1-python-on-windows/iris_data_5.py`

```
import os

import pandas as pd

import settings_1

DATA_ROOT = os.path.join(settings_1.PROJ_ROOT, "data")

df = pd.read_csv(f"{DATA_ROOT}/Iris.csv")

print(df.shape)
```

- Loading Iris Dataset using Pandas function with Path variable from package:
  `1-python-on-windows/config_1/settings_2.py`

```
import os
from pathlib import Path
```

```

def get_proj_root():
    return (
        # os.path.dirname(Path(__file__).resolve())
        # TODO: update project root path
        os.path.dirname(Path(__file__).resolve().parent)
        if "__file__" in locals() or "__file__" in globals()
        # NOTICE: The current working directory should be the project root folder in Python Terminal
        else os.path.dirname(os.path.realpath("__file__"))
    )


PROJ_ROOT = get_proj_root()
DATA_ROOT = os.path.join(PROJ_ROOT, "data")
```

`1-python-on-windows/iris_data_6.py`

```
import pandas as pd

from config_1 import settings_2

df = pd.read_csv(f"{settings_2.DATA_ROOT}/Iris.csv")

print(df.shape)
```

- Loading Iris Dataset using Pandas function with URL:

`1-python-on-windows/iris_data_7.py`

```
import pandas as pd

df = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
)

print(df.shape)
```

(2) Making `src` folder to save source code:
Multiple Packages (Folders) and Multiple Modules (Files) within src folder

- Loading Iris Dataset using scikit-learn function:

`1-python-on-windows/src/iris_data_1.py`

```
from sklearn import datasets

iris = datasets.load_iris()

X = iris.data
y = iris.target

print(X.shape)
print(y.shape)
```

- Loading Iris Dataset using Pandas function:

`1-python-on-windows/src/iris_data_2.py`

```
import pandas as pd

# NOTICE: FileNotFoundError: [Errno 2] No such file or directory: 'data/Iris.csv'
#           if the current working directory is in src/
```

```
#           or if the mode is interactive mode (Python Terminal or Interactive Window)
df = pd.read_csv("data/Iris.csv")

print(df.shape)
```

- Loading Iris Dataset using Pandas function with **Path variable**:

`1-python-on-windows/src/iris_data_3.py`

```
import os
from pathlib import Path

import pandas as pd

PROJ_ROOT = None

# script mode
if "__file__" in locals() or "__file__" in globals():
    # PROJ_ROOT = os.path.dirname(Path(__file__).resolve())
    # TODO: update project root path
    PROJ_ROOT = os.path.dirname(Path(__file__).resolve().parent)
# interactive mode
else:
    # NOTICE: The current working directory should be the project root folder in Python Terminal
    PROJ_ROOT = os.path.dirname(os.path.realpath("__file__"))

DATA_ROOT = os.path.join(PROJ_ROOT, "data")

df = pd.read_csv(f"{DATA_ROOT}/Iris.csv")

print(df.shape)
```

- Loading Iris Dataset using Pandas function with **Path variable from function**:

`1-python-on-windows/src/iris_data_4.py`

```

import os
from pathlib import Path

import pandas as pd


def get_proj_root():
    """Returns the path of project root directory

    Returns:
        str: text sequence type (strings) of the path of project root directory
    """
    return (
        # os.path.dirname(Path(__file__).resolve())
        # TODO: update project root path
        os.path.dirname(Path(__file__).resolve().parent)
        if "__file__" in locals() or "__file__" in globals()
        # NOTICE: The current working directory should be the project root folder in Python Terminal
        else os.path.dirname(os.path.realpath("__file__"))
    )


PROJ_ROOT = get_proj_root()
DATA_ROOT = os.path.join(PROJ_ROOT, "data")
```

```
27 df = pd.read_csv(f"{DATA_ROOT}/Iris.csv")
28 print(df.shape)
```

- Loading Iris Dataset using Pandas function with **Path variable from** **module**:

`1-python-on-windows/src/settings_3.py`

```
1  import os
2  from pathlib import Path
3
4
5  def get_proj_root():
6      """Returns the path of project root directory
7
8      Returns:
9          str: text sequence type (strings) of the path of project root directory
10     """
11     return (
12         # os.path.dirname(Path(__file__).resolve())
13         # TODO: update project root path
14         os.path.dirname(Path(__file__).resolve().parent)
15         # os.path.dirname(Path(__file__).resolve().parents[1])
16         if "__file__" in locals() or "__file__" in globals()
17         # NOTICE: The current working directory should be the project root folder in
       Python Terminal
18         else os.path.dirname(os.path.realpath("__file__"))
19     )
20
21
22 PROJ_ROOT = get_proj_root()
23
24 if __name__ == "__main__":
25     print(f"PROJ_ROOT: {PROJ_ROOT}")
```

`1-python-on-windows/src/iris_data_5.py`

```
2  import os
3
4  import pandas as pd
5
6  # NOTICE: ONLY in Python Terminal
7  # FIXME: ModuleNotFoundError: No module named 'settings_3'
8  # -> TODO: set PYTHONPATH environment variable
9  import settings_3
10
11 DATA_ROOT = os.path.join(settings_3.PROJ_ROOT, "data")
12
13 df = pd.read_csv(f"{DATA_ROOT}/Iris.csv")
14
15 print(df.shape)
```

TODO: Setting **PYTHONPATH** variable in **VS Code**

Use of the PYTHONPATH variable

> The `PYTHONPATH` environment variable specifies additional locations where the Python interpreter should look for modules. In VS Code, `PYTHONPATH` can be set through the terminal settings `(terminal.integrated.env.*)` and/or within an `.env` file.

```
File > Preferences > Settings
> Search settings: "terminal.integrated.env"
> Terminal > Integrated > Env : Windows > Edit in settings.json
> Terminal > Integrated > Env : Linux > Edit in settings.json
> Terminal > Integrated > Env : Osx > Edit in settings.json
```

`settings.json` : `"PYTHONPATH": "${env:PYTHONPATH}:${workspaceFolder}/src"`

```
{
    "terminal.integrated.env.linux": {
        "PYTHONPATH": "${env:PYTHONPATH}:${workspaceFolder}/src"
    },
    "terminal.integrated.env.osx": {
        "PYTHONPATH": "${env:PYTHONPATH}:${workspaceFolder}/src"
    },
    "terminal.integrated.env.windows": {
        "PYTHONPATH": "${env:PYTHONPATH};${workspaceFolder}/src"
    }
}
```

TODO: Setting **PYTHONPATH** variable in **.env**

> An example of when to use `PYTHONPATH` would be if you have source code in a `src` folder and tests in a `tests` folder. When running tests, however, those tests can't normally access modules in `src` unless you hard-code relative paths.
>
> To solve this problem, you could add the path to `src` to `PYTHONPATH` by creating an `.env` file within your VS Code workspace.

```
PYTHONPATH=./src
```

TODO: **Close and Reopen** the Python Project folder with **VS Code** (NOT Reload Window)

`1-python-on-windows/python_info.bat`

```
python -c "import sys; print(sys.executable)"

python -c "import sys; print(sys.path)"

echo %PYTHONPATH%

python -c "import os; print(os.getcwd())"
```

- Loading Iris Dataset using Pandas function with **Path variable from** **package**:

`1-python-on-windows/src/config_2/settings_4.py`

```
1  import os
2  from pathlib import Path
3
4  from dotenv import load_dotenv
5
6
7  def get_proj_root():
8      """Returns the path of project root directory
9
10     Returns:
11         str: text sequence type (strings) of the path of project root directory
12     """
13     return (
14         # os.path.dirname(Path(__file__).resolve())
```

```
        # TODO: update project root path
        # os.path.dirname(Path(__file__).resolve().parent.parent)
        os.path.dirname(Path(__file__).resolve().parents[1])
        if "__file__" in locals() or "__file__" in globals()
        # NOTICE: The current working directory should be the project root folder in Python Terminal
        else os.path.dirname(os.path.realpath("__file__"))
    )


PROJ_ROOT = get_proj_root()

DOTENV_PATH = os.path.join(PROJ_ROOT, ".env")
if os.path.isfile(DOTENV_PATH):
    # load .env file and set them as environment variables
    load_dotenv(DOTENV_PATH)

# Setting PYTHONPATH if not exists
if not os.getenv("PYTHONPATH"):
    os.environ["PYTHONPATH"] = os.path.join(PROJ_ROOT, "src")

SRC_ROOT = os.getenv("PYTHONPATH", os.path.join(PROJ_ROOT, "src"))

DATA_ROOT = os.path.join(PROJ_ROOT, "data")
MODEL_ROOT = os.path.join(PROJ_ROOT, "models")

if __name__ == "__main__":
    print(f"PROJ_ROOT: {PROJ_ROOT}")
    print(f"SRC_ROOT: {SRC_ROOT}")
    print(f"DATA_ROOT: {DATA_ROOT}")
    print(f"MODEL_ROOT: {MODEL_ROOT}")
```

`1-python-on-windows/src/iris_data_6.py`

```
import pandas as pd

# NOTICE: ONLY in Python Terminal
# FIXME: ModuleNotFoundError: No module named 'config_2'
# -> TODO: set PYTHONPATH environment variable
from config_2 import settings_4

df = pd.read_csv(f"{settings_4.DATA_ROOT}/Iris.csv")

print(df.shape)
```

Environment variable definitions file - **One Source Multi Use (OSMU)**

An environment variable definitions file is a simple text file containing key-value pairs in the form of `environment_variable=value`, with `#` used for comments. Multiline values are not supported, but values can refer to any other environment variable that's already defined in the system or earlier in the file. For more information, see **Variable substitution**. Environment variable definitions files can be used for scenarios such as debugging and tool execution (including linters, formatters, IntelliSense, and testing tools), but are not applied to the terminal.

By default, the Python extension looks for and loads a file named `.env` in the current workspace folder, then applies those definitions. The file is identified by the default entry `"python.envFile": "$workspaceFolder/.env"` in your user settings (see **General settings**). You can change the `python.envFile` setting at any time to use a different definitions file.

> For example, when developing a web application, you might want to easily switch between development and production servers. Instead of coding the different URLs and other settings into your application directly, you could use separate definitions files for each. For example:

`1-python-on-windows/dev.env`:

```
# dev.env - development configuration

# API endpoint
MYPROJECT_APIENDPOINT=https://my.domain.com/api/dev/

# Variables for the database
MYPROJECT_DBURL=https://my.domain.com/db/dev
MYPROJECT_DBUSER=devadmin
MYPROJECT_DBPASSWORD=!dfka**213=
```

`1-python-on-windows/prod.env`:

```
# prod.env - production configuration

# API endpoint
MYPROJECT_APIENDPOINT=https://my.domain.com/api/

# Variables for the database
MYPROJECT_DBURL=https://my.domain.com/db/
MYPROJECT_DBUSER=coreuser
MYPROJECT_DBPASSWORD=kKKfa98*11@
```

- Loading Iris Dataset using Pandas function with URL:
  `1-python-on-windows/src/iris_data_7.py`

```
import pandas as pd

df = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
)

print(df.shape)
```

1.1.3 Debugging (Removing Errors); Fixing Syntax Errors and Logic Errors

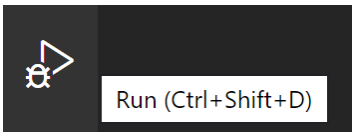
Debug = **De (Remove)** + **Bug (Errors)**
Python Debugging in VS Code

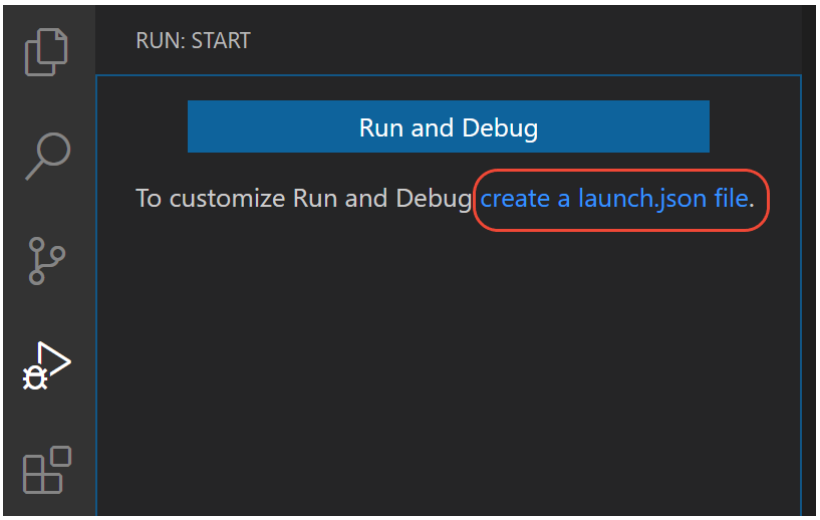
Python Debug Configurations in Visual Studio Code

- Initialize configurations

(1)

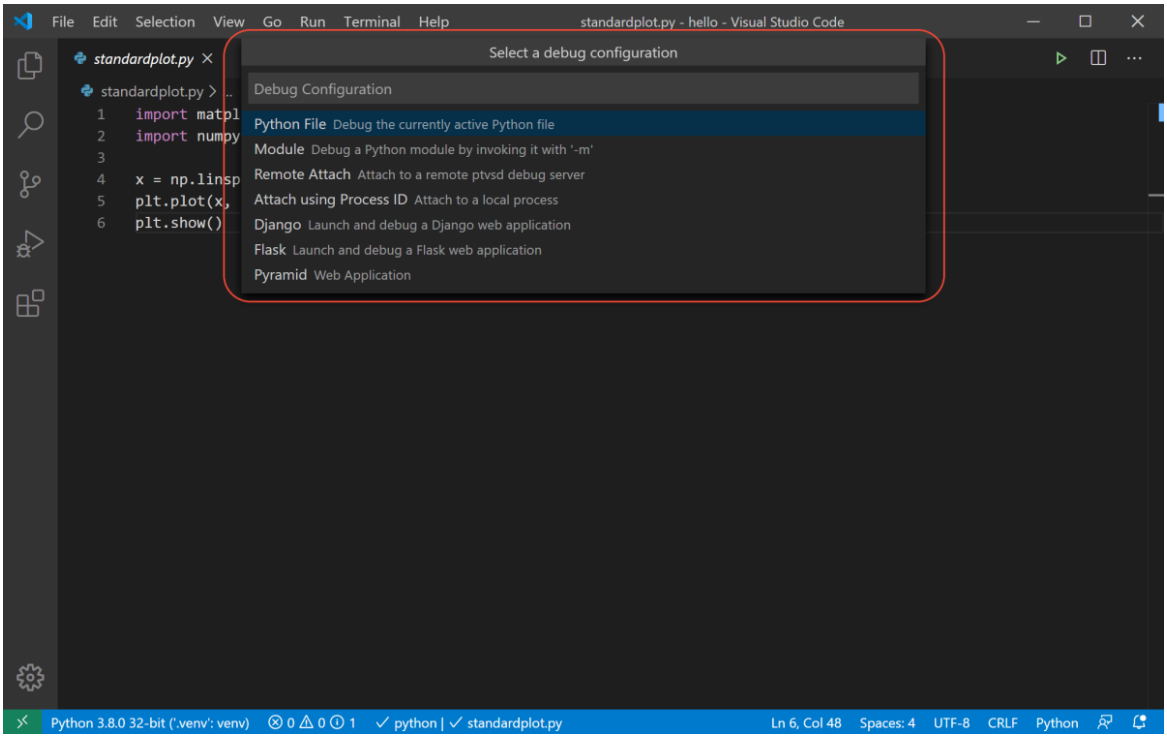
> Click the `create a launch.json file` link (circled in the image above) or use the `Run > Open configurations` menu command.





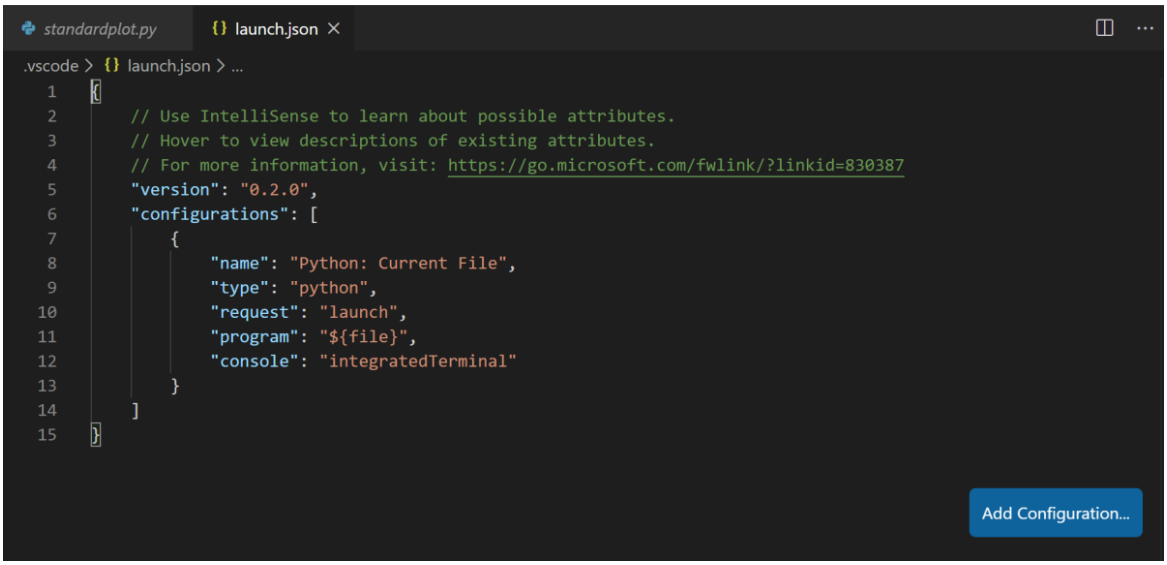
(2)

A configuration menu will open from the Command Palette allowing you to choose the type of debug configuration you want for the opened file. For now, in the `Select a debug configuration` menu that appears, select `Python File`.



(3)

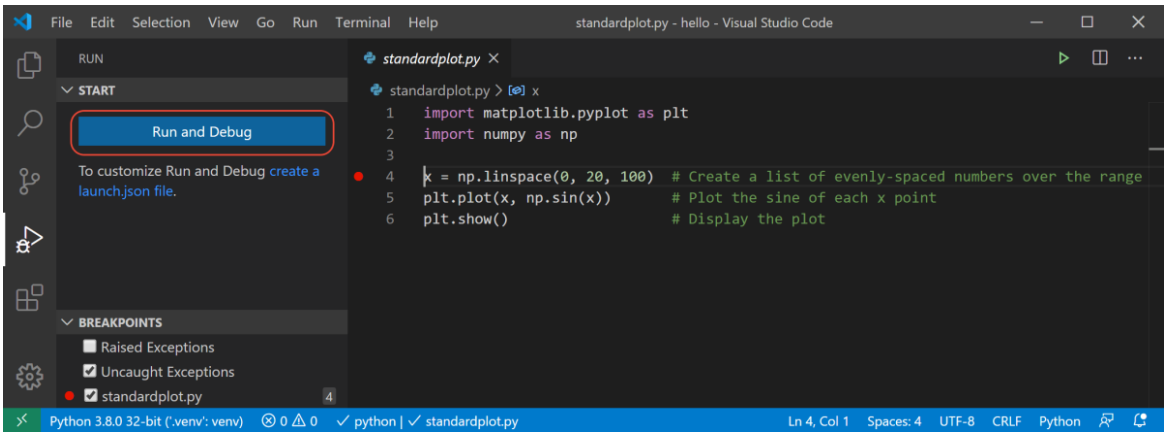
The Python extension then creates and opens a `launch.json` file that contains a pre-defined configuration based on what you previously selected, in this case, `Python File`. You can modify configurations (to add arguments, for example), and also add custom configurations.



- Basic Debugging

(1)

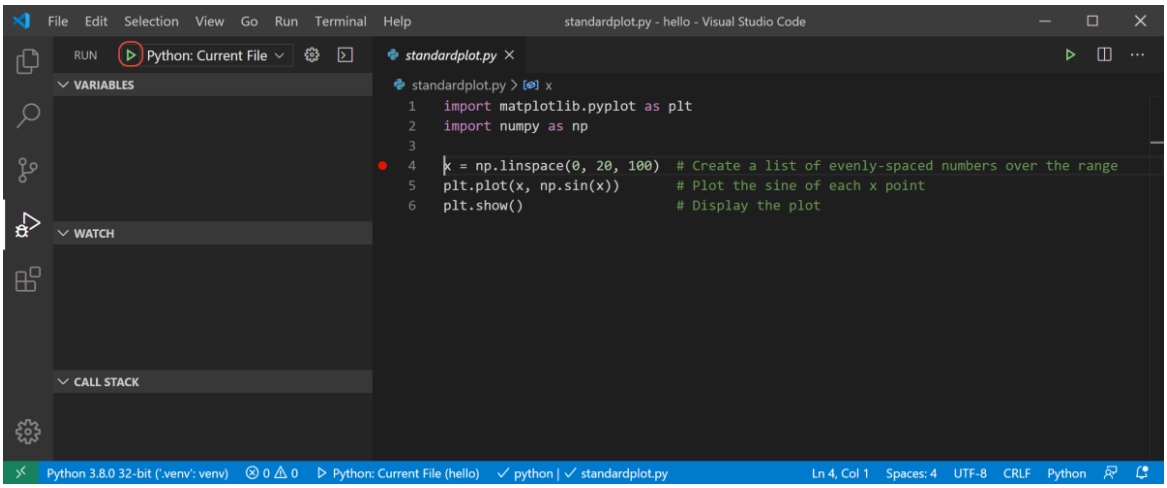
The simplest way to begin debugging a Python file is to use the `Run` view and click the `Run and Debug` button. When no configuration has been previously set, you will be presented with a list of debugging options. Select the appropriate option to quickly begin debugging your code.



(2)

Two common options are to use the `Python File` configuration to run the currently open Python file or to use the `Attach using Process ID` configuration to attach the debugger to a process that is already running.

For information about creating and using debugging configurations see the **Initialize configurations** and **Additional configurations** sections. Once a configuration is added, it can be selected from the dropdown list and started using the `Start Debugging` button.



Debugging `1-python-on-windows/src/iris_data_6.py`

```
2  import pandas as pd
3
4  # NOTICE: ONLY in Python Terminal
5  # FIXME: ModuleNotFoundError: No module named 'config_2'
6  # -> TODO: set PYTHONPATH environment variable
7  from config_2 import settings_4
8
9  df = pd.read_csv(f"{settings_4.DATA_ROOT}/Iris.csv")
10
11 print(df.shape)
```

(1) Setting a breakpoint (`F9`) at **line 7**

(2) Initializing the debugger (`F5`)

(3) Running the debugger and continuing the program to the end
Configure and run the debugger

> A debug toolbar appears along the top with the following commands from left to right:
>
> continue (`F5`), step over (`F10`), step into (`F11`), step out (`Shift+F11`), restart (`Ctrl+Shift+F5`), and stop (`Shift+F5`).

1.1.4 Testing to Find Errors

(1) Installing a Test Framework

(2) Creating Tests

(3) Enabling a Test Framework

(4) Discovering Tests (Test Discovery)

(5) Running Tests

Python testing in Visual Studio Code

(1) Installing a Test Framework

pytest: simple powerful testing with Python

```
py -3 -m pip install pytest
```

(2) Creating Tests

Tests outside application code

Putting tests into an extra directory outside your actual application code might be useful if you have many functional tests or for other reasons want to keep tests separate from actual application code (often a good idea).

`1-python-on-windows/tests/test_settings.py`:

```
1 from settings import get_proj_root
2
3
4 def test_get_proj_root():
5     assert (
6         get_proj_root()
7         == "C:\\Users\\tad\\workspaces\\issues-in-computer-programming\\1-python-on-
    windows"
8     )
```

`1-python-on-windows/tests/test_config_settings.py`:

```
1 from config.settings import get_proj_root
2
3
4 def test_get_proj_root():
5     assert (
6         get_proj_root()
7         == "C:\\Users\\tad\\workspaces\\issues-in-computer-programming\\1-python-on-
    windows"
8     )
```

(3) Enabling a Test Framework

```
The Command Palette (Ctrl+Shift+P or F1)
> Python: Configure Tests
> Select a test framework/tool to enable > pytest - pytest framework
> Select the directory containing the tests > tests
```

Enable a test framework

Testing in Python is disabled by default. To enable testing, use the `Python: Configure Tests` command on the Command Palette. This command prompts you to select a test framework, the folder containing tests, and the pattern used to identify test files.

(4) Discovering Tests (Test Discovery)

- Configuring Test Discovery

```
File > Preferences > Settings
> Search settings: "autoTestDiscoverOnSaveEnabled"
> Python > Testing > Auto Test Discover On Save Enabled
> [ ] Enable auto run test discovery when saving a test file.

File > Preferences > Settings
> Search settings: "python testing"
> Python > Testing: Pytest Enabled
> [X] Enable testing using pytest.
```

- Discovering Tests using ”Python: Discover Tests” command on the Command Palette

```
The Command Palette (Ctrl+Shift+P or F1)
> Python: Discover Tests
```

VS Code uses the currently enabled testing framework to discover tests. You can trigger test discovery at any time using the `Python: Discover Tests` command.

`python.testing.autoTestDiscoverOnSaveEnabled` is set to `true` by default, meaning test discovery is performed automatically whenever you save a test file. To disable this feature, set the value to `false`.

Test discovery applies the discovery patterns for the current framework (which can be customized using the **Test configuration settings**). The default behavior is as follows:

- `python.testing.unittestArgs`: Looks for any Python (`.py`) file with ”test” in the name in the top-level project folder. All test files must be importable modules or packages. You can customize the file matching pattern with the `-p` configuration setting, and customize the folder with the `-t` setting.
- `python.testing.pytestArgs`: Looks for any Python (`.py`) file whose name begins with ”test_” or ends with ”_test”, located anywhere within the current folder and all subfolders.

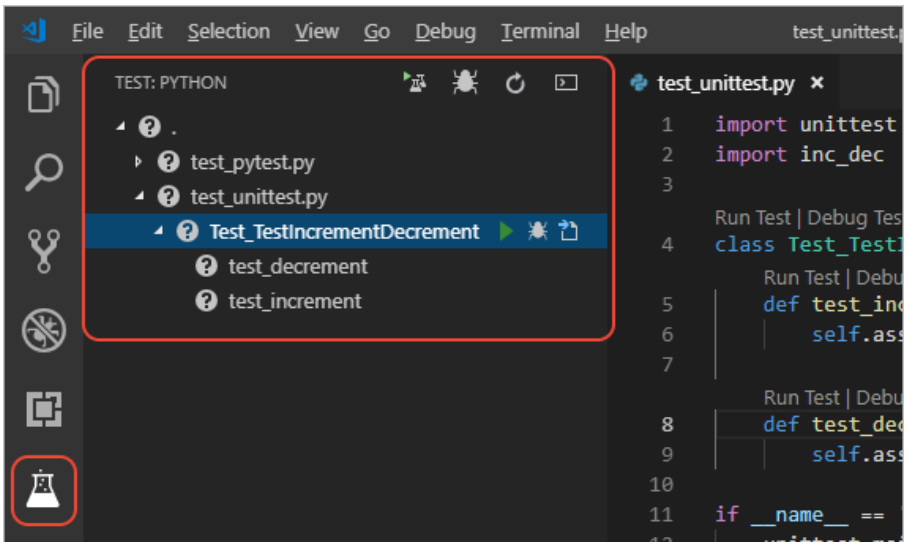
If discovery succeeds, the status bar shows Run Tests instead:

If discovery fails (for example, the test framework isn’t installed), you see a notification on the status bar. Selecting the notification provides more information:

(5) Running Tests

Running Tests using the Test Explorer on the VS Code activity bar

For Python, test discovery also activates the `Test Explorer` with an icon on the VS Code activity bar. The `Test Explorer` helps you visualize, navigate, and run tests:



1.1.5 Miscellaneous Issues

`1-python-on-windows/src/iris_data_analysis.py`:

- Loading Data from File using Package and Module

```
29 import pandas as pd
30
31 from config import settings
32
33 df = pd.read_csv(f"{settings.DATA_ROOT}\Iris.csv")
34
35 print(df.shape)
```

- Saving and Loading Scaler for Test Dataset

```
299 import pathlib
300 import pickle
301
302 # from config import settings
303 from config.settings import MODEL_ROOT
304
305 # Creating MODEL_ROOT directory if not exists
306 pathlib.Path(MODEL_ROOT).mkdir(parents=True, exist_ok=True)
307
308 SCALE_FILE = f"{MODEL_ROOT}/iris_scaler.pkl"
309
310 # pickle.dump(scaler, open(SCALE_FILE, "wb"))
311 with open(SCALE_FILE, "wb") as file:
312     pickle.dump(scaler, file)
313
314 # scaler2 = pickle.load(open(SCALE_FILE, "rb"))
315 with open(SCALE_FILE, "rb") as file:
316     scaler2 = pickle.load(file)
317
318 print(type(scaler2))
319
320 X_test = scaler2.transform(X_test)
```

- Saving and Loading Model

```
523 import pickle
524
```

```python
from config.settings import MODEL_ROOT

MODEL_FILE = f"{MODEL_ROOT}/iris_logistic.pkl"

# pickle.dump(classifier, open(MODEL_FILE, "wb"))
with open(MODEL_FILE, "wb") as file:
    pickle.dump(classifier, file)

# classifier2 = pickle.load(open(MODEL_FILE, "rb"))
with open(MODEL_FILE, "rb") as file:
    classifier2 = pickle.load(file)

print(type(classifier2))

print(f"Test score: {classifier2.score(X_test, y_test) * 100} %")
```

- i(N)terpreting Data, Visualizing Data
  TODO: run the `src/iris_kmeans_dash.py`

```python
# ================================================================================
# NOTICE: i(N)terpreting Data, Visualizing Data
# ================================================================================

# Plotly Dash vs Streamlit – Which is the best library for building data dashboard web apps?
# https://towardsdatascience.com/plotly-dash-vs-streamlit-which-is-the-best-library-for-building-data-dashboard-web-apps-97d7c98b938c

# Gradio vs Streamlit vs Dash vs Flask
# https://towardsdatascience.com/gradio-vs-streamlit-vs-dash-vs-flask-d3defb1209a2

# Streamlit vs Dash vs Voilà vs Panel – Battle of The Python Dashboarding Giants
# https://medium.datadriveninvestor.com/streamlit-vs-dash-vs-voil%C3%A0-vs-panel-battle-of-the-python-dashboarding-giants-177c40b9ea57

# Streamlit vs. Dash vs. Shiny vs. Voila vs. Flask vs. Jupyter
# https://datarevenue.com/en-blog/data-dashboarding-streamlit-vs-dash-vs-shiny-vs-voila

# TODO: Iris k-means clustering (iris_kmeans_dash.py)
```

1.2 For Single Python Project with Multiple Writers

1.2.1 (Running others' src code) Run Dependencies

Packages List to run the Python scripts

- `1-python-on-windows/src/iris_data_analysis.py`:

```
78 import matplotlib.pyplot as plt
79 import missingno as msno
80 import seaborn as sns
```

  - jupyter
  - pytest
  - pandas
  - matplotlib and seaborn
  - **...**

requirements.txt for Packages Installation

- `1-python-on-windows/requirements.txt`

```
pytest
jupyter
pandas
missingno
numpy
scipy
matplotlib
seaborn
scikit-learn
plotly
dash
dash-bootstrap-components
```

- Requirement Specifiers

```
SomeProject
SomeProject == 1.3
SomeProject >=1.2,<2.0
SomeProject[foo, bar]
SomeProject~=1.4.2
```

- Using Python environments in VS Code

> **Tip:** When you're ready to deploy the application to other computers, you can create a `requirements.txt` file with the command `pip freeze > requirements.txt` (`pip3` on macOS/Linux). The requirements file describes the packages you've installed in your virtual environment. With only this file, you or other developers can restore those packages using `pip install -r requirements.txt` (or, again, `pip3` on macOS/Linux). By using a requirements file, you need not commit the virtual environment itself to source control.

```
py -3 -m pip freeze > requirements.txt
py -3 -m pip install -r requirements.txt
```

Disadvantages:

- including **unnecessary packages** if **Anaconda Interpreter** is pre-installed

- jupyter dependencies: necessary packages vs. dependencies

```
pipdeptree > deps.txt
```

`1-python-on-windows/deps.txt:`

```
jupyter==1.0.0
  - ipykernel [required: Any, installed: 6.2.0]
    - debugpy [required: >=1.0.0,<2.0, installed: 1.4.1]
    - ipython [required: >=7.23.1,<8.0, installed: 7.26.0]
      - backcall [required: Any, installed: 0.2.0]
      - colorama [required: Any, installed: 0.4.4]
      - decorator [required: Any, installed: 5.0.9]
      - jedi [required: >=0.16, installed: 0.18.0]
        - parso [required: >=0.8.0,<0.9.0, installed: 0.8.2]
      - matplotlib-inline [required: Any, installed: 0.1.2]
        - traitlets [required: Any, installed: 5.0.5]
          - ipython-genutils [required: Any, installed: 0.2.0]
      - pickleshare [required: Any, installed: 0.7.5]
      - prompt-toolkit [required: >=2.0.0,<3.1.0,!=3.0.1,!=3.0.0, installed: 3.0
    .19]
        - wcwidth [required: Any, installed: 0.2.5]
      - pygments [required: Any, installed: 2.9.0]
      - setuptools [required: >=18.5, installed: 57.0.0]
      - traitlets [required: >=4.2, installed: 5.0.5]
        - ipython-genutils [required: Any, installed: 0.2.0]
```

Recommended Requirements File Formats

- `1-python-on-windows/requirements/py-base.txt`

```
# Python 2 and 3 compatibility utilities
# https://pypi.org/project/six/
six==1.16.0

# Command line utility to show dependency tree of packages
# https://pypi.org/project/pipdeptree/
pipdeptree==2.0.0

# Read key-value pairs from a .env file and set them as environment variables
# https://pypi.org/project/python-dotenv/
python-dotenv==0.17.1
```

- `1-python-on-windows/requirements/ds-base.txt`

```
-r py-base.txt

# Jupyter metapackage. Install all the Jupyter components in one go.
# https://pypi.org/project/jupyter/
jupyter==1.0.0

# Powerful data structures for data analysis, time series, and statistics
# https://pypi.org/project/pandas/
pandas==1.2.4

# Missing data visualization module for Python.
# https://pypi.org/project/missingno/
missingno==0.4.2

# NumPy is the fundamental package for array computing with Python.
# https://pypi.org/project/numpy/
numpy==1.20.3
```

```
# SciPy: Scientific Library for Python
# https://pypi.org/project/scipy/
scipy==1.6.3

# Python plotting package
# https://pypi.org/project/matplotlib/
matplotlib==3.4.2

# seaborn: statistical data visualization
# https://pypi.org/project/seaborn/
seaborn==0.11.1

# A set of python modules for machine learning and data mining
# https://pypi.org/project/scikit-learn/
scikit-learn==0.24.2

# An open-source, interactive data visualization library for Python
# https://pypi.org/project/plotly/
plotly==4.14.3

# A Python framework for building reactive web-apps. Developed by Plotly.
# https://pypi.org/project/dash/
dash==1.20.0

# Bootstrap themed components for use in Plotly Dash
# https://pypi.org/project/dash-bootstrap-components/
dash-bootstrap-components==0.12.2
```

1.2.2 (Readability) Code Formatter

What if many people are writing code in different styles? punctuation

Python Formatters: `black`

- Formatter-specific settings

```
py -3 -m pip install black
```

Configuring VS Code for Python Formatting on Save

```
File > Preferences > Settings
> Search settings: "python formatting provider"
> Python > Formatting: Provider
> [black]

File > Preferences > Settings
> Search settings: "format on save"
> Editor: Format On Save
> [X] Format a file on save.
```

Python Sort Imports: `isort`

- Python Sort Imports in VS Code

> Sort Imports uses the `isort` package to consolidate specific imports from the same module into a single `import` statement and to organize `import` statements in alphabetical order.

```
# NOTICE: pylint depends on isort
py -3 -m pip install pylint
```

Configuring VS Code for Python Sort Imports

```
File > Preferences > Settings
> Search settings: "python.sortImports.args"
> Python > Sort Imports: Args
> Add Item > "-rc" > OK
> Add Item > "--atomic" > OK
```

1.2.3 (Reasoning) Commenting & Documenting Code for Code Explanation

- Why to Comment and Document
  - ”Code is more often read than written.” — Guido van Rossum
  - ”Code tells you how; Comments tell you why.” — Jeff Atwood (aka Coding Horror)
  - ”It doesn’t matter how good your software is, because if the documentation is not good enough, people will not use it.” — Daniele Procida

Python Comments

Commenting Code (Commenting Single Line)

Commenting code using the pound sign ‘`#`‘ for Multiple Purposes (`Ctrl + /`)

- Code Description; Algorithmic Description
- Planning and Reviewing
- **Tagging**
  - VS Code Extension, TODO Highlight: **# TODO:**, **# FIXME:**, **# NOTICE:**

```
File > Preferences > Settings
> Search settings: "TODO Highlight"
> Todohighlight: Keywords > Edit in settings.json
```

`%USERPROFILE%/AppData/Roaming/Code/User/settings.json:`

```
{
    "todohighlight.keywords": [
        "NOTICE:",
    ],
}
```

Documenting Code (Commenting Multiple Lines) (vs. Multi-Line # with shortcut)

Documenting Code using Docstrings with the triple-double quote ‘`"""`‘ string format
PEP 257 – Docstring Conventions

- Three Major Categories
  - Package and Module Docstrings: Functions < Modules (Files) < Packages (Folders)
  - Script Docstrings: Script and functions

    - Class Docstrings: Class and class methods
- Docstrings related Packages
    - lazydocs: docstrings ⟶ Markdown
      `1-python-on-windows/docs/4-apis/md/index.md`
    - pdoc3: docstrings ⟶ html
      `1-python-on-windows/docs/4-apis/html/src/index.html`

1.2.4 Documenting a Python Project for Large Scale

Documenting a Python Project in `1-python-on-windows/docs/` folder

Modeling with Tools ⟶ `1-python-on-windows/docs/3-models/`

- Why to Model
    - Table of Contents for writing a Book
    - Blue Print for Complex and Complicated System: Department of Defense (DoD); Pentagon
    - Design enables better software
- How to Model
    - Modulation and Composition
    - The C4 model for visualising software architecture:
      Context ⟶ Containers ⟶ Components ⟶ Code
    - Modeling Tools: Diagrams as Code/Text, PlantUML
      PlantUML Prerequisites:
        - Java (Open JDK Downloads for Windows x86_64)
        - VS Code Extension (jebbs.plantuml)

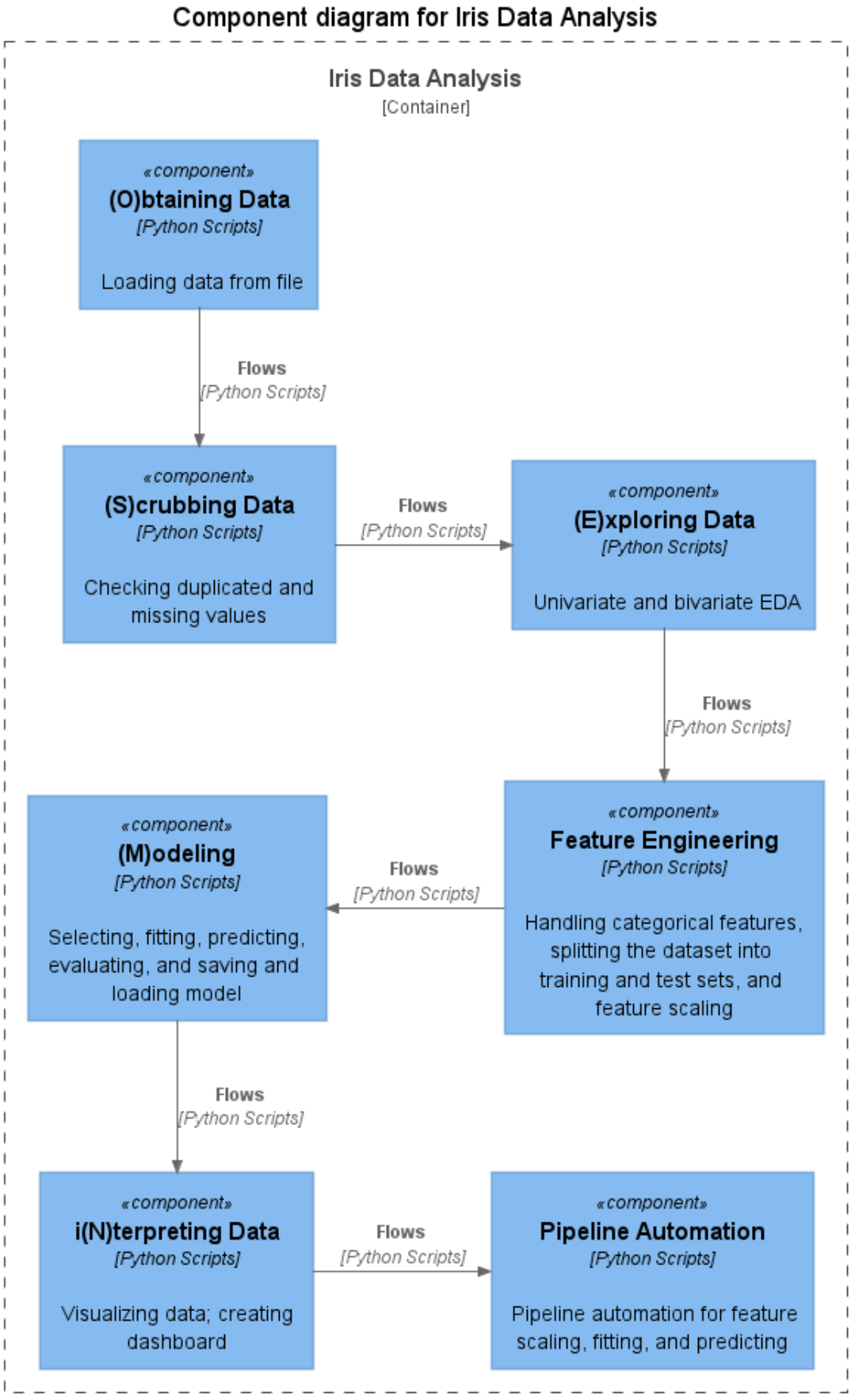
      You DON'T need to install the PlantUML if you use VS Code Extension, jebbs.plantuml.
      Configuring VS Code for PlantUML:

```
File > Preferences > Settings
> Search settings: "Plantuml: Render"
> Plantuml: Render
> Local

File > Preferences > Settings
> Search settings: "Plantuml: Java"
> Plantuml: Java
> Java executable location:
  C:\Program Files\ojdkbuild\java-1.8.0-openjdk-1.8.0.275-1\bin\java.exe
```

If you want to use **PlantUML in LaTeX**, execute below in command prompt.

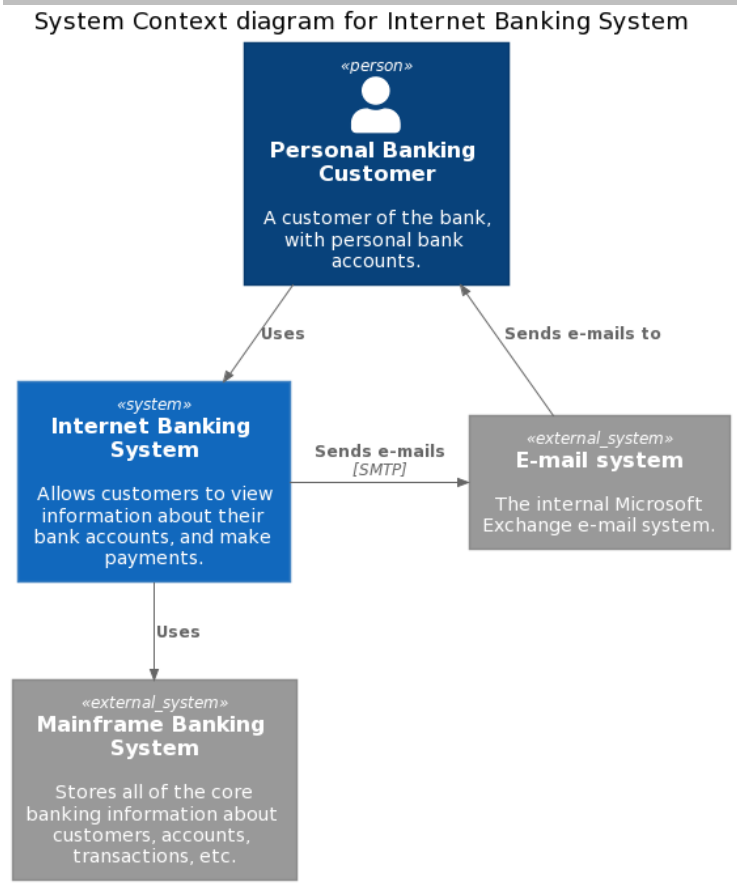
```
:: plantuml - A LuaLaTeX package for PlantUML in LaTeX
:: https://koppor.github.io/plantuml/
:: https://github.com/koppor/plantuml

:: TODO: run this batch file as Administrator

dir %USERPROFILE% /s /b | findstr /i plantuml.jar > plantuml_path.txt
set /p PLANTUML_JAR_PATH=<plantuml_path.txt
del /q plantuml_path.txt

set PLANTUML_JAR=%PLANTUML_JAR_PATH%
setx /m PLANTUML_JAR %PLANTUML_JAR_PATH%
```

- Component diagram for Iris Data Analysis:
  `1-python-on-windows/docs/3-models/Iris_C4-3_Component.puml`

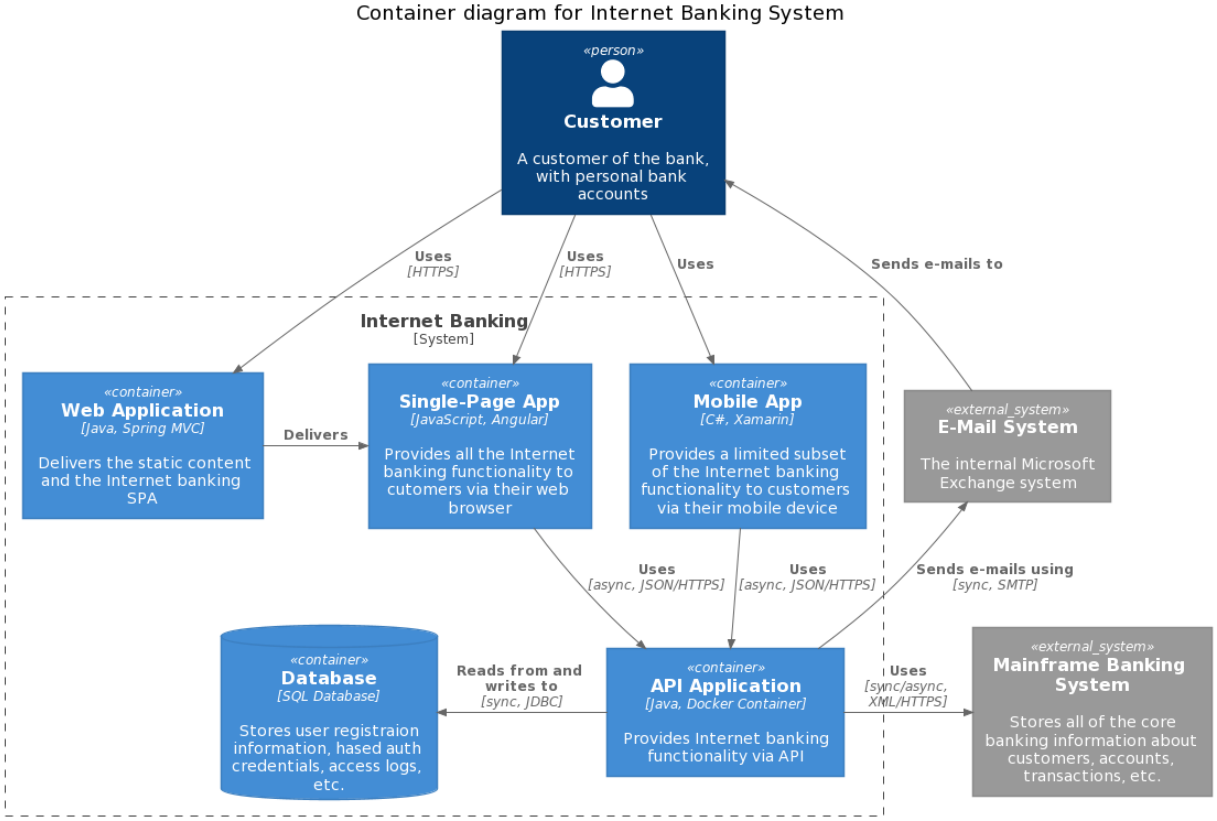


- The C4 model for Internet Banking System

`1-python-on-windows/docs/3-models/C4-1_Context_Diagram_Sample-bigbankplc.puml`

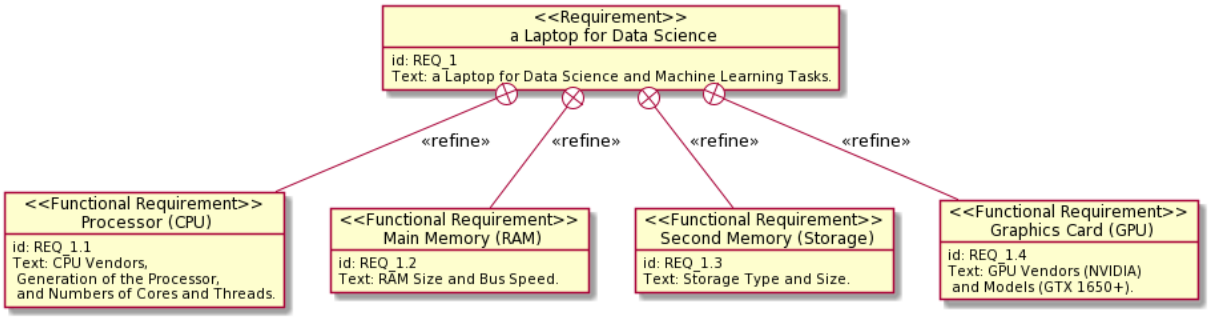


`1-python-on-windows/docs/3-models/C4-2_Container_Diagram_Sample-bigbankplc.puml`



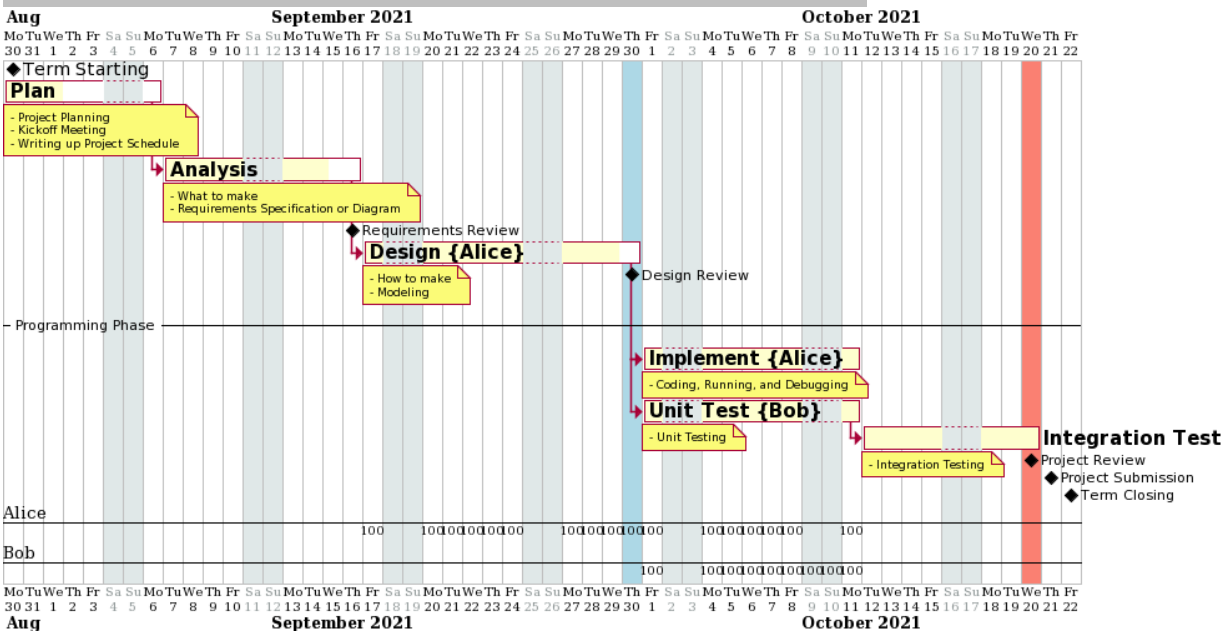
Requirements Specification ⟶ `1-python-on-windows/docs/2-spec/`

- Modeling Requirements with SysML
- Requirements Specification for Data Science Laptop
  `1-python-on-windows/docs/2-spec/ds_laptop_req.puml`



Planning ⟶ `1-python-on-windows/docs/1-plan/`

- Gantt Diagram
  Gantt chart with PlantUML: Term Project Schedule
  `1-python-on-windows/docs/1-plan/project-schedule-term.puml`



README.md: `1-python-on-windows/README.md`

- Project Overall Description

1.3 For Multiple Python Projects

Multiple Projects in a Single Computer

1.3.1 Python Runtime Issues

Python Interpreter Version and Package Issues:
pip doesn't allow to install different versions of package at the same time

- Python 3.6 and Package Versions
  - pandas <= 1.1.5
  - numpy <= 1.19.5
  - scipy <= 1.5.4
- Python 3.9 and Package Versions
  - pandas 1.2.4
  - numpy 1.20.3
  - scipy 1.6.3

1.3.2 Python Multiple Runtimes and Environments

Project Dependent **Python Version and Packages** **in a Project Folder**

Python Multiple Runtimes

- Install Python 3 interpreters **from lowest to highest**
  - Python 3.6
  - Python 3.7
  - Python 3.8
  - Python 3.9

Python Global and Virtual Environments (venv)

Using Python environments in VS Code

By default, any Python interpreter that you've installed runs in its own **global environment**, which is not specific to any one project. For example, if you just run `python` (Windows) or `python3` (macOS/Linux) at a new command prompt, you're running in that interpreter's global environment. Accordingly, any packages that you install or uninstall affect the global environment and all programs that you run within that context.

Although working in the global environment is an easy way to get started, that environment will, over time, become cluttered with many different packages that you've installed for different projects. Such clutter makes it difficult to thoroughly test an application against a specific set of packages with known versions, which is exactly the kind of environment you'd set up on a build server or web server.

For this reason, developers often create a **virtual environment** for a project. A virtual environment is a subfolder in a project that contains a copy of a specific interpreter. When you activate the virtual environment, any packages you install are installed only in that environment's subfolder. When you then run a Python program within that environment, you know that it's running against only those specific packages. Be aware that if you're not using a **virtual environment**, and you have multiple versions of Python installed and set in the `path` environment variable, you might need to specify the Python interpreter to use in the terminal for installing packages to the global environment.

- Working with virtual environments

(1) Creating a virtual environment **in the project root folder** **with Python version**

```
py -3.9 -m venv .venv
```

(2) **Activating the virtual environment**, otherwise Python packages will be installed in the Global Environment

```
.venv\Scripts\activate
```

(3) Installing Python packages

```
pip install -r requirements/ds-base.txt
```

(4) Selecting a Python interpreter in VS Code

```
The Command Palette (Ctrl+Shift+P or F1)
> python: select interpreter
> Python 3.9.x 64-bit
```

1.3.3 pipdeptree for resolving package dependency conflicts in the same virtual environment

Numpy Example by Scikit-learn

- pipdeptree

```
pipdeptree > deps.txt
```

1-python-on-windows/deps.txt:

```
scikit-learn==0.24.2
  - joblib [required: >=0.11, installed: 1.0.1]
  - numpy [required: >=1.13.3, installed: 1.20.3]
  - scipy [required: >=0.19.1, installed: 1.6.3]
    - numpy [required: >=1.16.5,<1.23.0, installed: 1.20.3]
  - threadpoolctl [required: >=2.0.0, installed: 2.2.0]
```

Package Conflicts Example

(1) Package Conflicts Example with Python 3.6 and packages for Python 3.9

```
py -3.6 -m venv .venv
.venv\Scripts\activate
pip install -r requirements/ds-base.txt
```

1.4 Automation for Programming Python on Windows

1.4.1 Batch Scripts to only Install once Required Software on Windows

`1-python-on-windows/setups/`
Windows commands

(1) `1-sys-deps.bat`

- Installing Windows Package Manager: Chocolatey
  INSTALLING CHOCOLATEY, Search Chocolatey Packages

```
23 @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
       InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.
       Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET
       "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

(2) `2-py-runtimes.bat`

- Installing Python Runtime on Windows using Chocolatey
  Python (Chocolatey Package)

```
11 choco install -y python --version 3.6.8
12 choco install -y python --version 3.7.9
13 choco install -y python --version 3.8.10
14 choco install -y python --version 3.9.5
```

(3) `3-vscode.bat`

- Installing VS Code using Chocolatey
  Visual Studio Code (Chocolatey Package)

```
17 choco install -y vscode
```

- Installing VS Code Extensions for Python Development
  Command line extension management

```
31 code ^
32 --install-extension ms-python.vscode-pylance ^
33 --install-extension donjayamanne.python-extension-pack ^
34 --install-extension tushortz.python-extended-snippets ^
35 --install-extension KevinRose.vsc-python-indent ^
36 --install-extension oderwat.indent-rainbow ^
37 --install-extension almenon.arepl ^
38 --install-extension njpwerner.autodocstring ^
39 --install-extension wayou.vscode-todo-highlight ^
40 --install-extension vscode-icons-team.vscode-icons ^
41 --install-extension byi8220.indented-block-highlighting ^
42 --install-extension 2gua.rainbow-brackets ^
43 --install-extension mikestead.dotenv ^
44 --install-extension zenor.makefile-creator ^
45 --install-extension joaompinto.vscode-graphviz ^
46 --install-extension jebbs.plantuml ^
47 --install-extension searKing.preview-vscode ^
48 --install-extension DavidAnson.vscode-markdownlint ^
49 --install-extension ms-vscode.Theme-MarkdownKit ^
50 --install-extension mdickin.markdown-shortcuts ^
51 --install-extension bierner.markdown-preview-github-styles ^
52 --install-extension hbrok.markdown-preview-bitbucket ^
53 --install-extension bierner.markdown-shiki ^
54 --install-extension goessner.mdmath ^
```

```
55 --install-extension bierner.markdown-emoji ^
56 --install-extension yzhang.markdown-all-in-one ^
57 --install-extension tomoki1207.pdf ^
58 --install-extension GrapeCity.gc-excelviewer ^
59 --install-extension mechatroner.rainbow-csv ^
60 --install-extension James-Yu.latex-workshop
```

(4) `_setup-all.bat`

- Installing all required software with just one batch file
  Windows Command: call

```
5 call 1-system-deps.bat
6 call 2-py-runtimes.bat
7 call 3-vscode.bat
```

1.4.2 Makefile for Building a Python Project repeatedly

`1-python-on-windows/Makefile`

- GNU Make
- Make for Windows

(1) Creating a virtual environment

- `make venv`

```
56 venv:
57   py -$(PY_VER) -m venv $(VENV)
```

(2) Activating the virtual environment

- `$(VENV_ACT)`

```
16 PY_VER := 3.9
17 VENV := venv-py$(PY_VER)
18 VENV_ACT := $(VENV)\Scripts\activate
```

(3) Installing Python packages

- `make deps`

```
59 deps:
60   $(VENV_ACT) \
61     && pip install -r requirements\prereqs.txt \
62     && pip install -r requirements\dev.txt \
63     && pip list \
64     && pipdeptree \
65     && pipdeptree --json > deps.json \
66     && pipdeptree > deps.txt
```

(4) Creating `.env` file if not exists

- `make .env`

```
68 .env:
69   if not exist .env \
70     echo PYTHONPATH=./src> %cd%\.env
```

(5) Running Python scripts

- `make run`

```
run:
  $(VENV_ACT) \
    && python src\iris_data_analysis.py \
    && python src\iris_kmeans_dash.py
```

(6) Testing Python scripts

- `make test`

```
test:
  $(VENV_ACT) \
    && pytest tests\test_settings.py \
    && pytest tests\test_config_settings.py
```

(7) Generating documentation files

- `make docs`

```
docs:
  make docs-md docs-tex docs-apis

docs-md:
  if not exist $(PROJ_DOCS)\pdfs \
    mkdir $(PROJ_DOCS)\pdfs
  for %%f in ($(PROJ_DOCS)\md\*.md) do \
    pandoc %%f --pdf-engine=xelatex -o %%f.pdf
  move $(PROJ_DOCS)\md\*.pdf $(PROJ_DOCS)\pdfs

docs-tex:
  if not exist $(PROJ_DOCS)\pdfs \
    mkdir $(PROJ_DOCS)\pdfs
  for %%f in ($(PROJ_DOCS)/tex/*.tex) do \
    pdflatex -output-directory=$(PROJ_DOCS)/pdfs $(PROJ_DOCS)/tex/%%f && make
    clean-latex

docs-apis:
  $(VENV_ACT) \
  && lazydocs \
    --output-path="docs/4-apis/md" \
    --overview-file="index.md" \
    src \
  && pdoc \
    --force \
    --html \
    --output-dir docs/4-apis/html \
    src
```

(8) Removing the virtual environment, documentation, and temporary folders

- `make clean`

```
clean:
  if exist $(VENV) \
    rmdir $(VENV) /s /q
  make clean-pdf clean-latex
  del *.pyc /s
  for /d /r . %%d in (.pytest_cache) do @if exist "%%d" rd /s/q "%%d"
  for /d /r . %%d in (.ipynb_checkpoints) do @if exist "%%d" rd /s/q "%%d"
  for /d /r . %%d in (__pycache__) do @if exist "%%d" rd /s/q "%%d"
```

(9) Executing all the tasks

- `make all`

```
29 TARGETS := clean clean-pdf clean-latex venv deps .env run test docs
30 .PHONY: all $(TARGETS)
31
32 all:
33   make $(TARGETS)
```

1.4.3 Automation in Source Code Level

- Data Science Pipeline Automation with sklearn.pipeline.Pipeline:
  `1-python-on-windows/src/iris_data_analysis.py`

```
570 —
571 import pickle
572
573 from sklearn import svm
574 from sklearn.linear_model import LogisticRegression
575 from sklearn.model_selection import train_test_split
576 from sklearn.pipeline import Pipeline
577 from sklearn.preprocessing import StandardScaler
578 from sklearn.tree import DecisionTreeClassifier
579
580 from config.settings import MODEL_ROOT
581
582 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
        random_state=0)
583
584 # ----------------------------------------
585 # Creating Pipelines, Model Fitting and Predicting
586 # ----------------------------------------
587
588 pipeline_lr = Pipeline(
589     [
590         ("std_scaler1", StandardScaler()),
591         ("lr_classifier", LogisticRegression(random_state=0)),
592     ]
593 )
594
595 pipeline_dt = Pipeline(
596     [("std_scaler2", StandardScaler()), ("dt_classifier", DecisionTreeClassifier())]
597 )
598
599 pipeline_svm = Pipeline(
600     [("std_scaler3", StandardScaler()), ("svm_classifier", svm.SVC())]
601 )
602
603 pipelines = [pipeline_lr, pipeline_dt, pipeline_svm]
604 pipe_dict = {0: "Logistic Regression", 1: "Decision Tree", 2: "Support Vector
        Machine"}
605
606 for pipe in pipelines:
607     pipe.fit(X_train, y_train)
608
609 for i, model in enumerate(pipelines):
610     print(f"{pipe_dict[i]} Test Accuracy: {model.score(X_test, y_test)}")
611
612 # ----------------------------------------
613 # Saving and Loading Models
614 # ----------------------------------------
615
```

```
616 MODEL_FILE = f"{MODEL_ROOT}/pipelines_models.pkl"
617
618 with open(MODEL_FILE, "wb") as file:
619     for pipe in pipelines:
620         pickle.dump(pipe, file)
621 #     pickle.dump(pipeline_lr, file)
622
623 pipelines2 = []
624 with open(MODEL_FILE, "rb") as file:
625     while True:
626         try:
627             pipelines2.append(pickle.load(file))
628         except EOFError:
629             break
630
631 for i, model in enumerate(pipelines2):
632     print(i, ":", model)
633     print(f"{pipe_dict[i]} Test Accuracy: {model.score(X_test, y_test)}")
```

Chapter 2

Programming Python on Linux

2.1 Why Linux for Data Science

- (Production Environments) Mostly Linux Servers
- (Development Environments) Linux only Data Science Software

2.1.1 (Production Environments) Mostly Linux Servers

Productionizing and Deploying Data Science Applications on Linux Servers

- How important is to learn Linux for a data scientist?

> **I would say it is very important to learn some Linux at some point throughout your data science journey.** Here are some reasons for this:
>
> - **In general, learning how to use a *NIX terminal will, at the very least, improve your productivity as a data scientist.** For example, package installation, environment set up, version control with Git, setting up your path "everything," etc., can be done in any terminal, and will probably get done on a Linux terminal once you move to the cloud.
> - **As a data scientist, it won't take long for you to come across datasets that will not fit into your computer's memory, and at that time, you will have to turn to solutions such as cloud computing.** It is very likely that, once this happens, you will use some flavour of Linux to spin up your cloud instance.
> - **Deep learning models benefit greatly from GPU's, and more specifically, NVIDIA GPU's.** Apple does not offer a computer with NVIDIA GPU's at the moment, hence, if you want to develop and test DL models in your computer before turning to the cloud, your best choice is to turn to **Linux or Windows on a computer with an NVIDIA graphics card**.
> - **NVIDIA also offers a great library called RAPIDS for working with massive datasets.** At the time of writing, this is **only available on some Linux distributions**.
> - **The applications you will help build will most likely be deployed on Linux.**

- Should You Pick Up Linux Skills For Data Science In 2021?

> **Where Linux Falls**
>
> With these technical requirements out of the way, I think it is now important to talk about where Linux falls in the industry these days.

> **Linux has a 98-percent market share in the server domain . That being said, if you are setting up a server, it is most likely going to be running Linux.** This is because the Linux kernel is not only more stable, but also has a lot more access to various commands along with their respective arguments and parameters from Bash.
>
> **Since most servers run on Linux, most of our endpoints are in turn also powered by Linux .** That being said, I think it is of vital importance to thoroughly understand a deployment topic prior to preparing to deploy. **The question I ask in this article is whether or not these skills are going to be a big selling point for a Data Scientist as a technical skill — and the answer to that question is yes .**
>
> Often this is not only implied, but also required by jobs. **Often Data Scientists need to deploy a model**, and I have been on teams where the deployment plan quickly falls apart. Fortunately, for most of these situations I have had the technical know-how when it comes to server administration to effectively continue our work and get the model working in any way we might need it to.

- Best Operating System OS for Data Science and Analytics

  1) It Doesn't Matter Which OS You Use.
  2) Programming Languages on the OS
  3) The Importance of Libraries in Data Science
  4) **Why You Should Consider Linux as Your OS .**

     > Simply put, it is far more flexible than any other OS. Originally developed by Linus Torvalds in the early 1990's. Torvalds was frustrated with the available or, unavailable options on the market. He decided to take MINIX, a operating system locked into the education system, and create his own operating system kernal, Linux Kernal.
     >
     > **Linux was built by a programmer for programmers. It is used by 80% or more of the data science field , and it is the leader in its ability to be customized.**
     >
     > **Most cloud servers are built on Linux.** Since most data is now being stored on Cloud servers this will streamline data deployment and accusation. If speed and usability are desirable to you then linux would be your choice.

  5) Resources to Help with your decision…

2.1.2 (Development Environments) Linux only Data Science Software

Computer Vision (CV)

- Major Computer Vision Problems
  The 5 Computer Vision Techniques That Will Change How You See The World,
  6 Significant Computer Vision Problems Solved by ML
  - Image Classification
  - **Object Localization and Detection**
  - **Object Tracking**
  - **Semantic and Instance Segmentation**
  - Data Generation using deep generative models such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs)
  - Neural Style Transfer
- Data Annotation Tools for Supervised Learning
  Labelme, LabelBox, LabelImg, VoTT (**Microsoft**), CVAT (**Intel**)
  - Computer Vision Annotation Tool (CVAT)

> CVAT is free, online, interactive **video and image annotation tool for computer vision**. It is being used by our team to annotate million of objects with different properties. Many UI and UX decisions are based on feedbacks from professional data annotation team.

New Computer Vision Tool Accelerates Annotation of Digital Images and Video

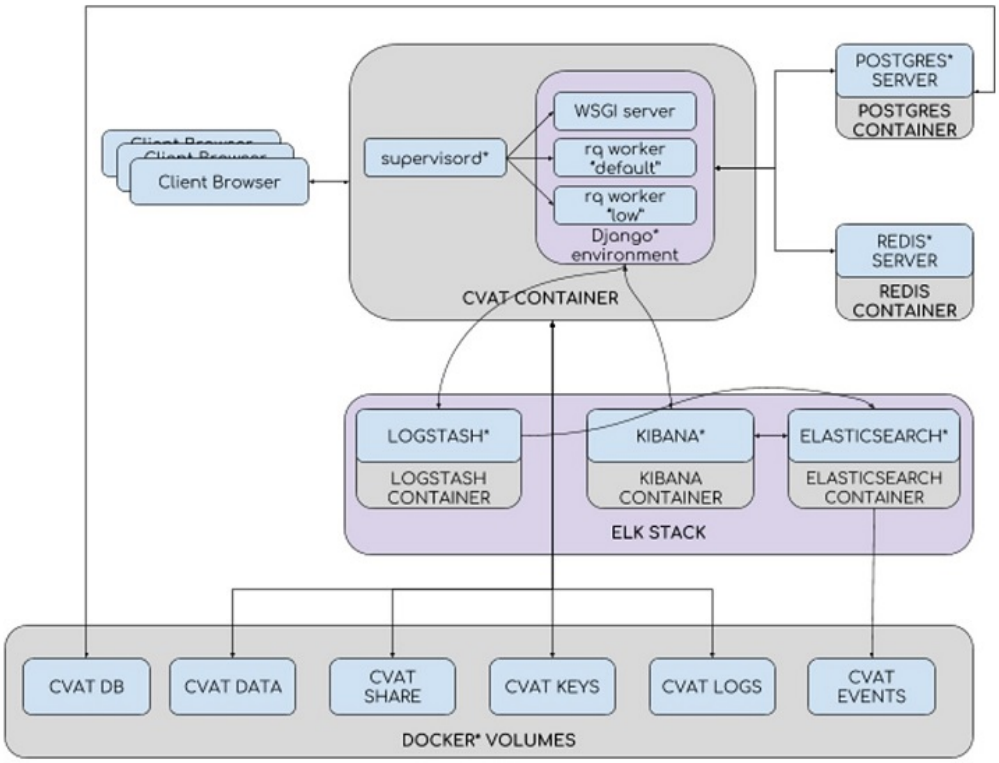
> we are pleased to present a new open source program called **Computer Vision Annotation Tool (CVAT, pronounced "si-vi-eɪ-ti")** that accelerates the process of **annotating digital images and videos** for use in training computer vision algorithms. In this post, we'll cover why CVAT is needed and CVAT's key capabilities. Please consult our article on **the Intel Developer Zone** for a detailed overview of how to use CVAT.

Installation Guide

> Run **docker containers**. It will take some time to download the latest CVAT release and other required images like postgres, redis, etc. from DockerHub and create containers.

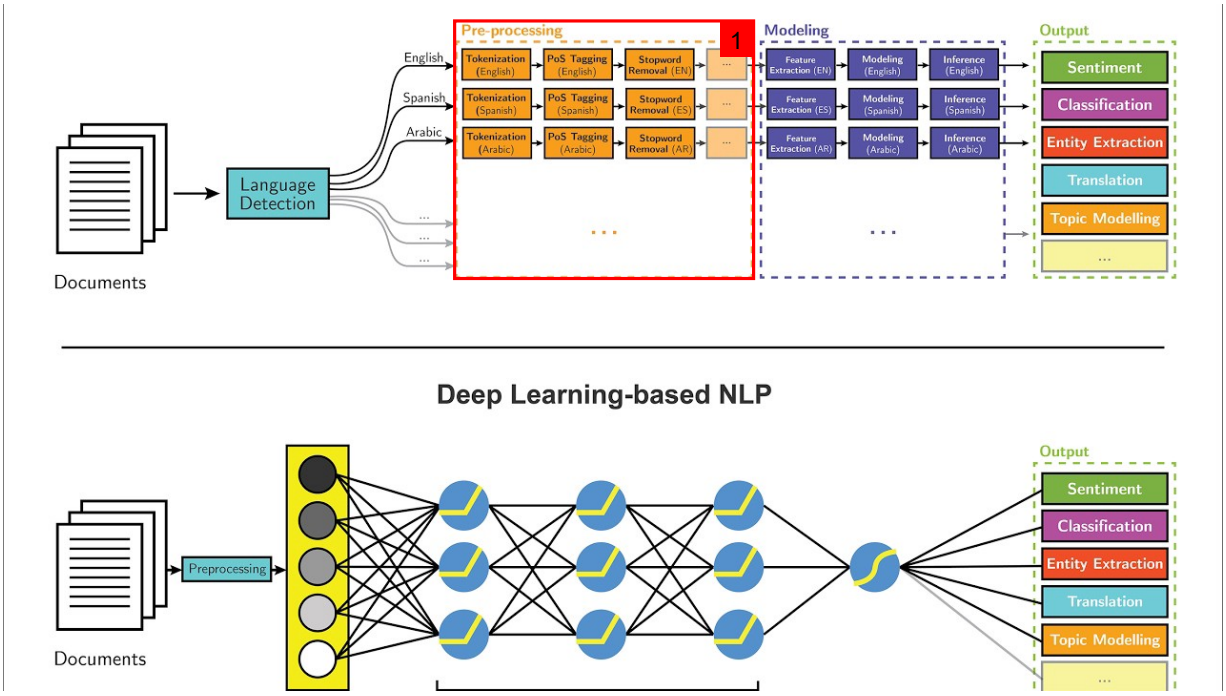
Computer Vision Annotation Tool: A Universal Approach to Data Annotation

The Internal Structure



Natural Language Processing (NLP); Nothing but Computational Linguistics

- Major NLP Problems
  The 7 NLP Techniques That Will Change How You Communicate in the Future (Part I),
  The 7 NLP Techniques That Will Change How You Communicate in the Future (Part II),
  Five Contemporary Natural Language Processing Problems Pushed Forward by DL



- Text Classification
- Entity Extraction; Named Entity Recognition (NER)
- Topic Modeling
- Sentiment Analysis
- Question Answering Systems (QAS)
- Text Summarization
- Text Generation
- Language Translation: Neural Machine Translation (NMT)

- Morphological Analysis and Part-of-Speech (POS) Tagging for Korean

  KKMA (Java),
  KOMORAN (Java),
  Open Korean Text (Java),
  Mecab-Ko (C++, JVM/Java/Scala),
  Arirang (Java),
  Rhino (R),
  Daon (Java),
  khaiii (C++),
  KoNLPy (Python wrapper),
  KoalaNLP (Scala wrapper)

  - Kakao Hangul Analyzer III (khaiii)

    khaiii stands for **Kakao Hangul Analyzer III** which is built by Kakao and the successor's name for **Daumkakao Hangul Analyzer II**.

    khaiii **DOES NOT** support **Microsoft Windows**.

GPU Data Science

- RAPIDS

RAPIDS

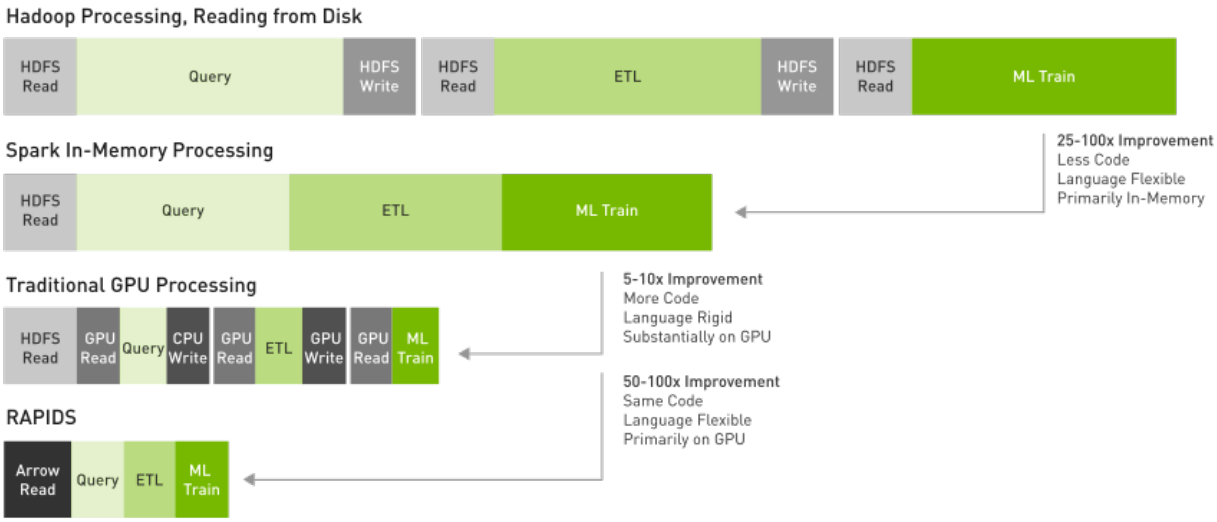
The **RAPIDS** suite of open source software libraries and APIs gives you the ability to execute end-to-end data science and analytics pipelines entirely on **GPUs**. Licensed under Apache 2.0, RAPIDS is incubated by **NVIDIA®** based on extensive hardware and data science experience. RAPIDS utilizes **NVIDIA CUDA®** primitives for low-level compute optimization, and exposes GPU parallelism and high-bandwidth memory speed through user-friendly Python interfaces.

Prerequisites:

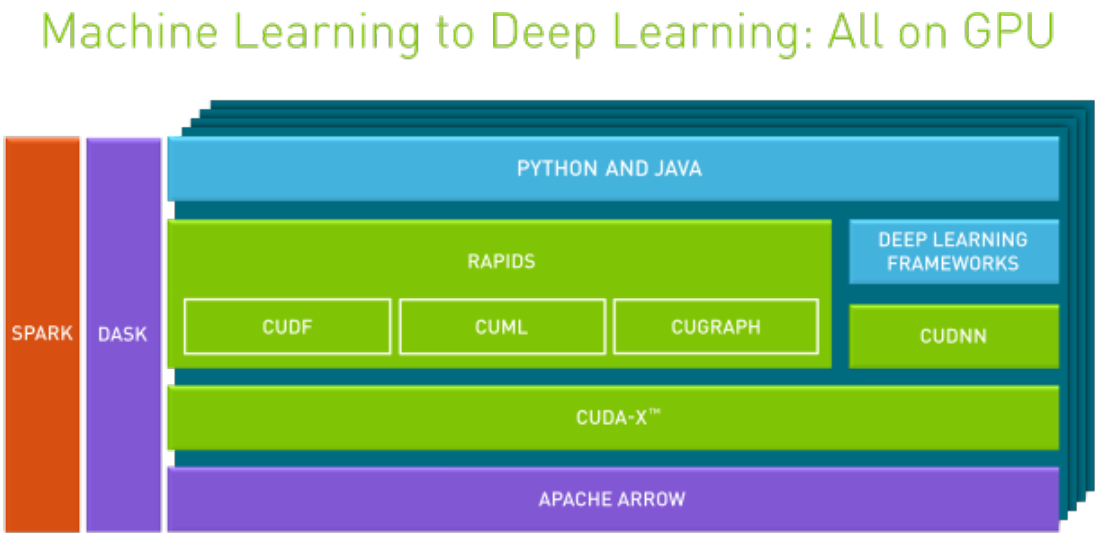
- OS: **Ubuntu** 16.04/18.04/20.04 or **CentOS** 7/8 with gcc/++ 9.0+

HIGH-PERFORMANCE DATA SCIENCE

Data Processing Evolution

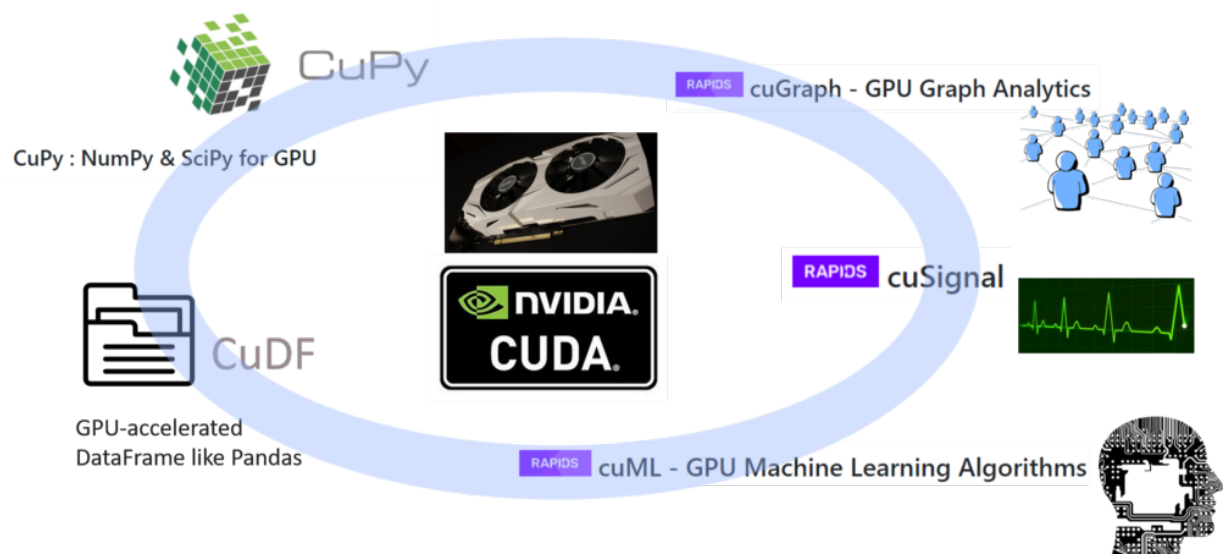


Machine Learning to Deep Learning: All on GPU



GPU-Powered Data Science (NOT Deep Learning) with RAPIDS

The fantastic RAPIDS ecosystem



- OpenAI, Triton

This is the development repository of **Triton, a language and compiler for writing highly efficient custom Deep-Learning primitives**. The aim of Triton is to provide an open-source environment to write fast code at higher productivity than CUDA, but also with higher flexibility than other existing DSLs.

Compatibility

Supported Platforms:

- **Linux**

Supported Hardware:

- NVIDIA GPUs (Compute Capability 7.0+)
- Under development: AMD GPUs, CPUs

Introducing Triton: **Open-Source GPU Programming** for Neural Networks,

OpenAI proposes open-source Triton language as **an alternative to Nvidia's CUDA**

2.2 Windows Subsystem for Linux (WSL)

2.2.1 WSL Concepts

Running Linux on Windows just like a Windows Application

- What is the Windows Subsystem for Linux?

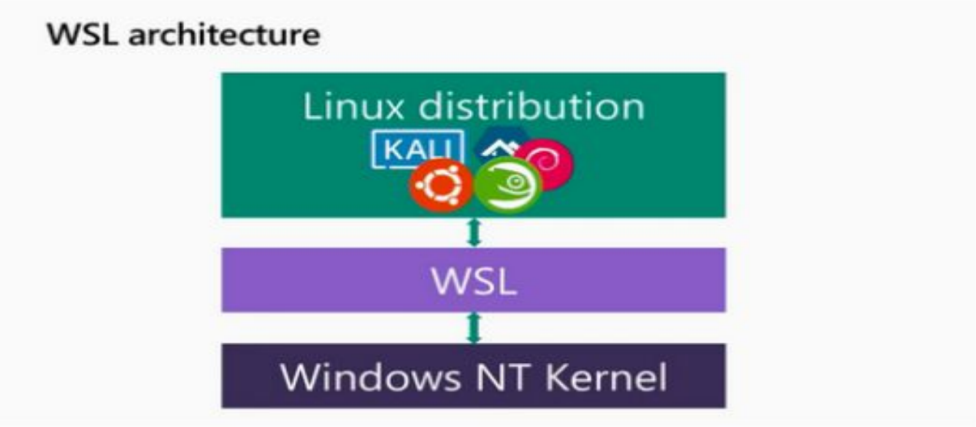
> The Windows Subsystem for Linux lets developers **run a GNU/Linux environment** – including most command-line tools, utilities, and applications – **directly on Windows**, unmodified, without the overhead of a traditional virtual machine or dualboot setup.
>
> **WSL 2 is a new version of the Windows Subsystem for Linux architecture** that powers the Windows Subsystem for Linux to run ELF64 Linux binaries on Windows. Its primary goals are to increase file system performance, as well as adding full system call compatibility.

Understanding WSL Architecture for WSL Setups

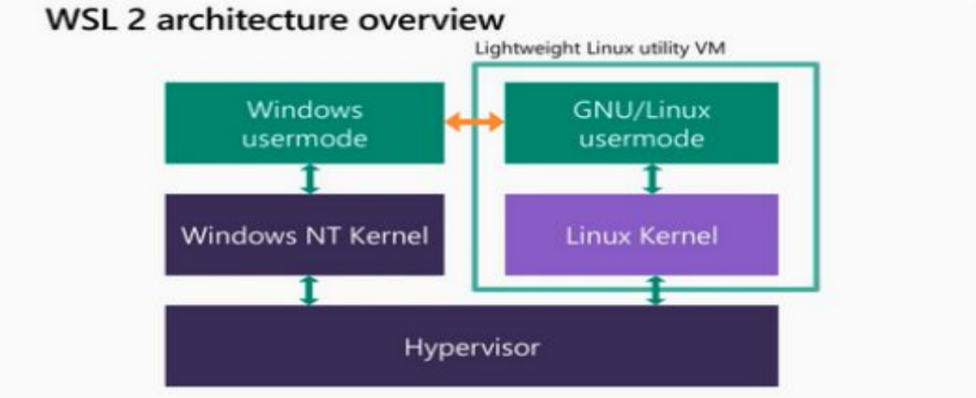
WSL2 for Developers, WSL 2: The new Windows Subsystem for Linux

> WSL 1 and the current version WSL 2 have a major difference in their architecture.
> **WSL 1** adds a **Linux compatibility-interface**, **'emulating' a Linux system** that you can work with. It supplies all the things the Linux user space needs to run. However, **all system calls (Linux ABIs) will still be handled by the Windows NT kernel** **by converting them in a translation layer**. This leads to longer response times for file I/O and a full Linux functionality cannot be offered. This is because Linux and Windows work fundamentally different in certain areas where a proper translation is not possible. In addition, new functionality on the Linux side needs some time to be also available within WSL 1 as it needs to be implemented first.
>
> **WSL2** changed this approach and **switched to a virtualization approach** instead. **A full Linux kernel is virtualized in the so-called** **light-weight utility VM based on Hyper-V** . Its design is based on the efforts made for Linux containers on Windows (LCOW) to minimize overhead and provide highest possible integration. The kernel itself is managed by Microsoft and especially optimized for the use within WSL 2. And all that is Open Source!

- WSL

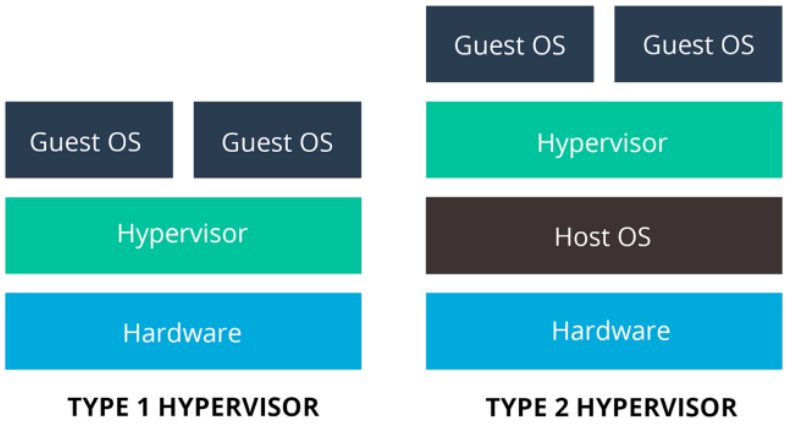


- WSL 2



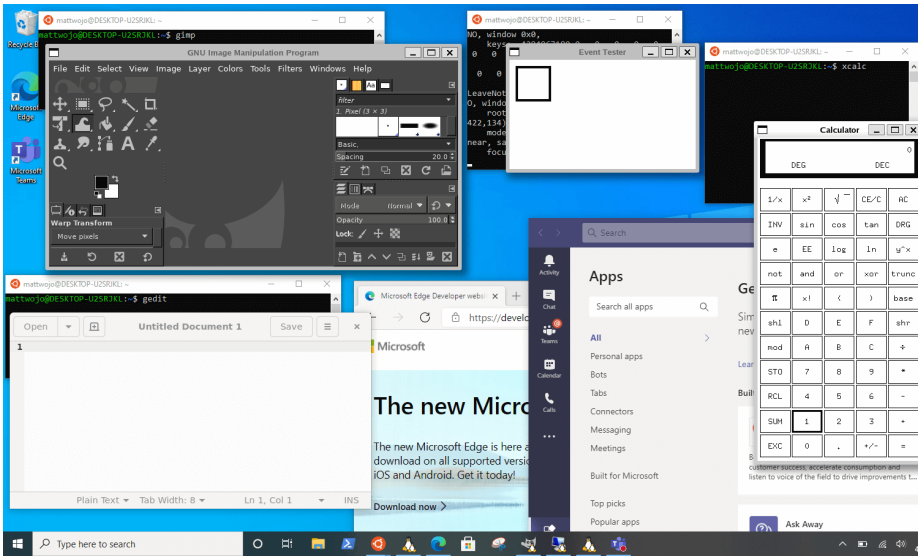
Introduction to Hyper-V on Windows 10

- Type 1 and Type 2 Hypervisors
  Type 1 and Type 2 Hypervisors: What Makes Them Different



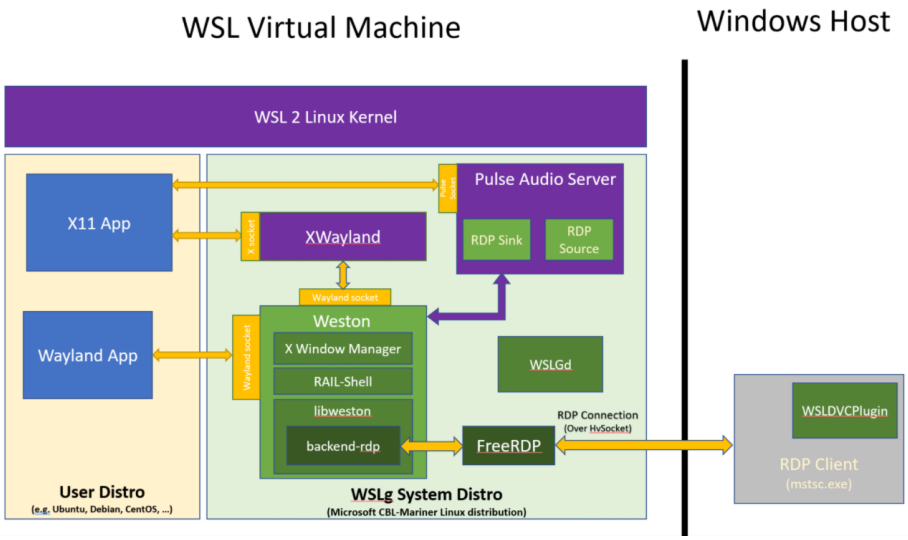
- GUI Applications Support in WSL
  Windows 11 could seamlessly run graphical Linux apps,
  Run Linux GUI apps on the Windows Subsystem for Linux (preview)

> You can now preview Windows Subsystem for Linux (WSL) **support for running Linux GUI applications (X11 and Wayland) on Windows** in a fully integrated desktop experience.
>
> WSL 2 enables Linux GUI applications to feel native and natural to use on Windows.



WSLg Architecture

> **GUI applications support** in the Windows Subsystem for Linux being available to **Windows Insiders**



2.2.2 WSL Setups

Enabling a Hardware Virtualization

Enabling **Intel VT and AMD-V** Virtualization Hardware Extensions in BIOS

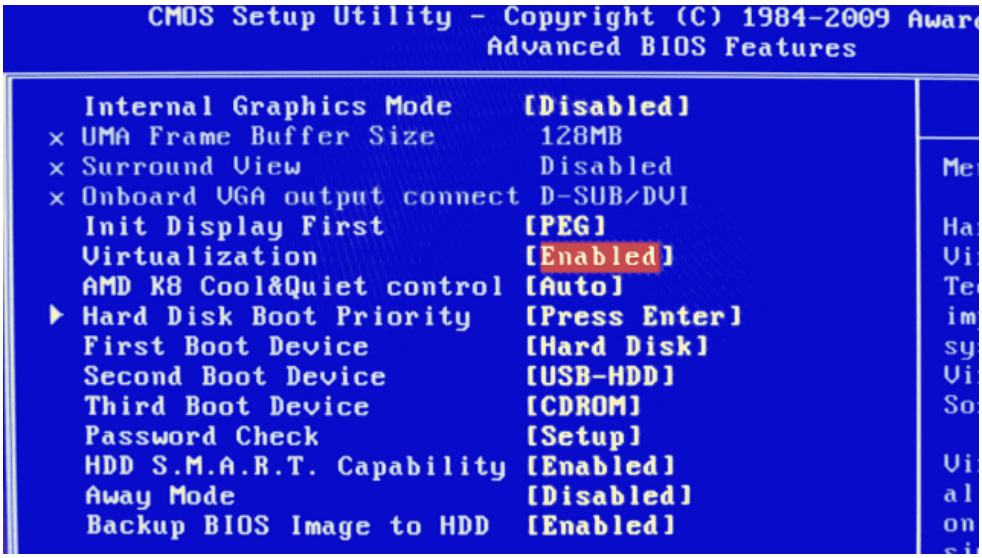
- Intel Virtual Technology (VT)
  How To Enable, Configure and Use Hyper-V on Windows 10

  > To get your computer prepared to run Hyper-V, make sure that the hardware virtualization support is enabled in the BIOS first. Boot into BIOS on your computer, enable **Virtualization Technology** under System Security. Depending on the version of BIOS you are running, you may need to poke around to find it.

- AMD Virtualization

How do I enable hardware virtualization technology (VT-x) for use in Virtualbox?

> Enter the BIOS (often pressing `Del` or `F12` while booting) and see with the manual how it is named there. Each BIOS appears to have a different name fror this. Search for **Virtualization, Virtualization Technology (VT-x), SVM, VMX**, or similar, here shown for an Award BIOS:



Setting up Hyper-V

Install Hyper-V on Windows 10

> The Hyper-V role **cannot be installed** on **Windows 10 Home**.

Open **Command Prompt as Administrator** and run:

- **TODO:** execute `2-python-on-linux/setups/windows/hyper-v/enable-hyper-v.bat`

```
11 powershell dism.exe /online /enable-feature /featurename:Microsoft-Hyper-V /all /
       norestart
```

Setting up WSL2 and Linux

Windows Subsystem for Linux Installation Guide for Windows 10

Open **Command Prompt as Administrator** and run:

(0) **TODO:** execute `2-python-on-linux/setups/windows/prerequisites.bat`

```
1  @echo off
2
3  pushd "%~dp0"
4
5  echo ############################################################################
6  echo Install Prerequisites
7  echo ############################################################################
8
9  set SCRIPT_HOME=%cd%
10
11 call "%SCRIPT_HOME%\prereqs\win-pkg-mgr.bat"
12
13 call "%SCRIPT_HOME%\prereqs\system-prereqs.bat"
```

2-python-on-linux/setups/windows/prereqs/**win-pkg-mgr.bat**

```
@echo off

echo ################################################################################
echo Windows Package Manager: chocolatey
echo ################################################################################

:: https://chocolatey.org/install

@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
    InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.
    WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%
    PATH%;%ALLUSERSPROFILE%\chocolatey\bin"

::powershell Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.
    ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::
    SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString
    ('https://chocolatey.org/install.ps1'))

choco upgrade -y chocolatey

:: How I Completely Automated Managing Windows Desktop Software...Forever
:: https://chocolatey.org/resources/case-studies/automated-managing-windows-desktop-
    software
```

2-python-on-linux/setups/windows/prereqs/**system-prereqs.bat**

```
@echo off

echo ################################################################################
echo Installing GNU related and other utilities for Windows
echo ################################################################################

echo ================================================================================
echo File Utilities
echo ================================================================================

echo --------------------------------------------------------------------------------
echo Get Files
echo --------------------------------------------------------------------------------

:: GNU Wget
:: https://chocolatey.org/packages/Wget
choco install -y wget --version 1.20

:: cURL
:: https://chocolatey.org/packages/curl
choco install -y curl

:: WinSCP (Install)
:: https://chocolatey.org/packages/winscp.install
choco install -y winscp.install

echo --------------------------------------------------------------------------------
echo Compress and Extract Files
echo --------------------------------------------------------------------------------

:: 7-Zip (Install)
:: https://chocolatey.org/packages/7zip.install
choco install -y 7zip.install
:: choco upgrade -y 7zip.install

:: Info-ZIP
```

```
:: https://chocolatey.org/packages/unzip
choco install -y unzip

:: TarTool
:: https://chocolatey.org/packages/tartool
choco install -y tartool

echo ------------------------------------------------------------------------
echo Find and Replace Text in Files
echo ------------------------------------------------------------------------

:: GNU grep
:: https://chocolatey.org/packages/grep
:: choco install -y grep

:: grepWin
:: https://chocolatey.org/packages/grepwin
:: choco install -y grepwin

:: GNU sed
:: https://chocolatey.org/packages/sed
choco install -y sed

echo ========================================================================
echo JSON Utilities
echo ========================================================================

:: jq
:: https://chocolatey.org/packages/jq
choco install -y jq

echo ========================================================================
echo Miscellaneous
echo ========================================================================

:: GNU Make
choco install -y make
```

(1) **TODO:** execute `2-python-on-linux/setups/windows/wsl/wsl-instl-1.bat`

NOTICE: This will restart your computer when prompted

```
@echo off

:: Windows Subsystem for Linux Installation Guide for Windows 10
:: https://docs.microsoft.com/en-us/windows/wsl/install-win10

echo ########################################################################
echo Windows Subsystem for Linux Installation Guide for Windows 10
echo ########################################################################

echo ========================================================================
echo Install Windows Subsystem for Linux
echo ========================================================================

echo ------------------------------------------------------------------------
echo Step 1 - Enable the Windows Subsystem for Linux
echo ------------------------------------------------------------------------

:: Open PowerShell as Administrator and run below
powershell dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem
    -Linux /all /norestart

```

```
echo ------------------------------------------------------------------------
echo Step 2 - Check requirements for running WSL 2
echo ------------------------------------------------------------------------

:: Check your Windows version (2004, Build 19041 or higher) by selecting the Windows Logo key + R, type winver, select OK
winver

echo ------------------------------------------------------------------------
echo Step 3 - Enable Virtual Machine feature
echo ------------------------------------------------------------------------

:: Open PowerShell as Administrator and run below
powershell dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

:: Restart your machine to complete the WSL install and update to WSL 2
shutdown /r /t 10
```

- Step 1 - Enable the Windows Subsystem for Linux
- Step 2 - Check requirements for running WSL 2
- Step 3 - Enable Virtual Machine feature

(2) **TODO:** execute `2-python-on-linux/setups/windows/wsl/wsl-instl-2.bat`

```
@echo off

:: Windows Subsystem for Linux Installation Guide for Windows 10
:: https://docs.microsoft.com/en-us/windows/wsl/install-win10

echo ########################################################################
echo Windows Subsystem for Linux Installation Guide for Windows 10
echo ########################################################################

echo ========================================================================
echo Install Windows Subsystem for Linux
echo ========================================================================

echo ------------------------------------------------------------------------
echo Step 4 - Download the Linux kernel update package
echo ------------------------------------------------------------------------

:: TODO: update WSL 2 Linux kernel
:: https://aka.ms/wsl2kernel

curl https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi -o %USERPROFILE%\Downloads\wsl_update_x64.msi
%USERPROFILE%\Downloads\wsl_update_x64.msi

echo ------------------------------------------------------------------------
echo Step 5 - Set WSL 2 as your default version
echo ------------------------------------------------------------------------

wsl --set-default-version 2

:: Set a distro to be backed by WSL 2
:: wsl --set-version Ubuntu-20.04 2

:: Verifying what versions of WSL your distro are using
:: wsl --list --verbose
:: wsl -l -v

```

```
37 echo ------------------------------------------------------------------------
38 echo Step 6 - Install your Linux distribution of choice
39 echo ------------------------------------------------------------------------
40
41 :: Microsoft Store > Search > "Ubuntu LTS" > Ubuntu 18.04 LTS
42 :: start https://www.microsoft.com/store/apps/9N9TNGVNDL3Q
43
44 :: TODO: install Ubuntu 20.04+ LTS using Microsoft Store
45 start https://www.microsoft.com/store/apps/9n6svws3rx71
```

- Step 4 - Download the Linux kernel update package
- Step 5 - Set WSL 2 as your default version
- Step 6 - Install your Linux distribution of choice: Installing **Ubuntu** **20.04 LTS**
  `Enter new UNIX username: ubuntu`

Why Ubuntu?

- Best Linux Distro For Machine Learning & AI: Comparison & Analysis

  > The Short Version Of The Answer
  >
  > **Distro#1: Ubuntu and its derivatives**
  >
  > Distro#2: Fedora and the RHEL family of distros

- Which Linux distro for data science?

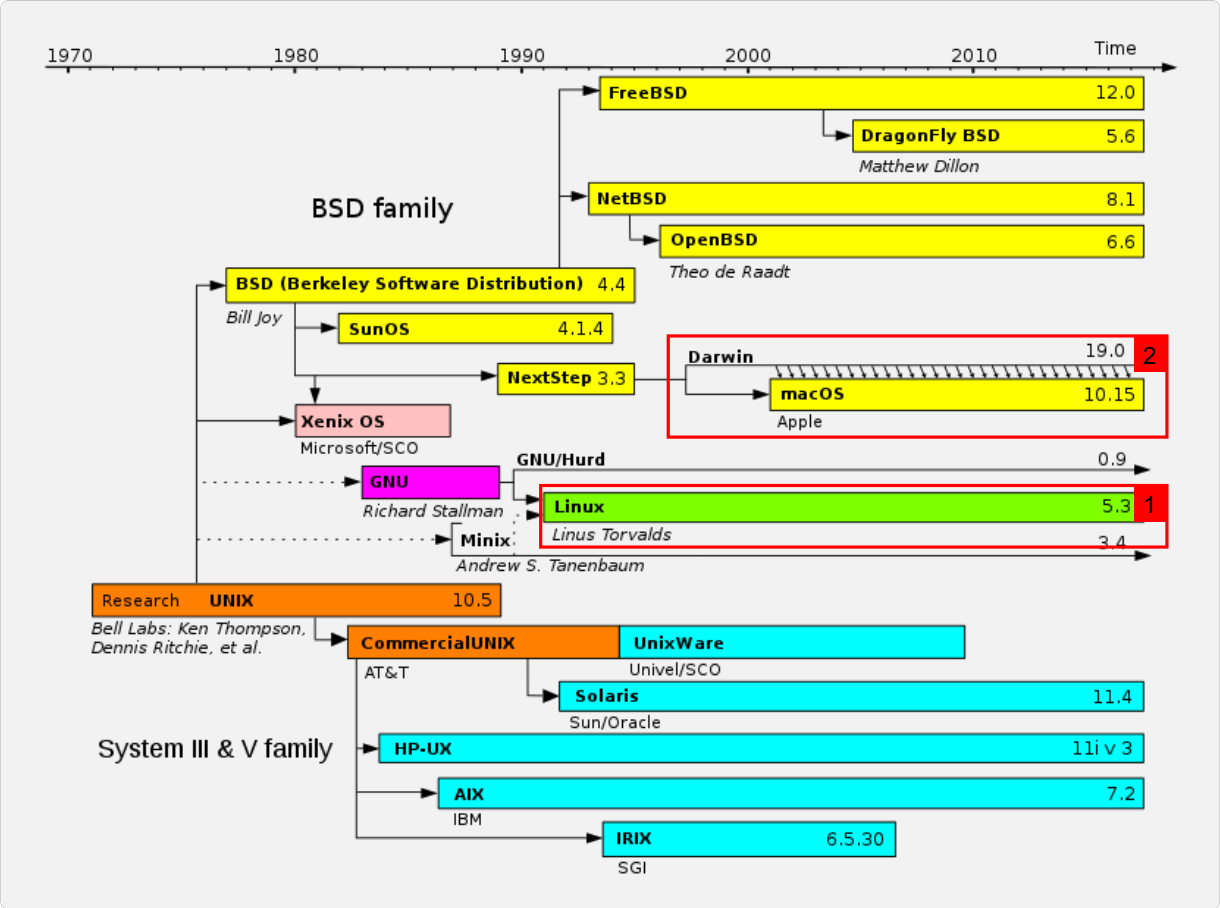
  > According to many articles on google (i.e. "https://www.whizlabs.com/blog/why-ubuntu-is-best-os-for-programming/"), there is no doubt that **the ubuntu is the best Linux distro for most programmers**. Thus, **I strongly recommend Ubuntu**.

- 7 Reasons Why Ubuntu is the Best OS for Programming

  1. Support for Emerging Technologies
  2. Access to Certified Hardware
  3. Consistent OS Experience
  4. Advantage of Streamlined Distribution
  5. Massive Support
  6. Upgrade Hardware without Any Restrictions
  7. Appealing User Interface

Unix Timeline and Linux Distribution Families

- Unix timeline

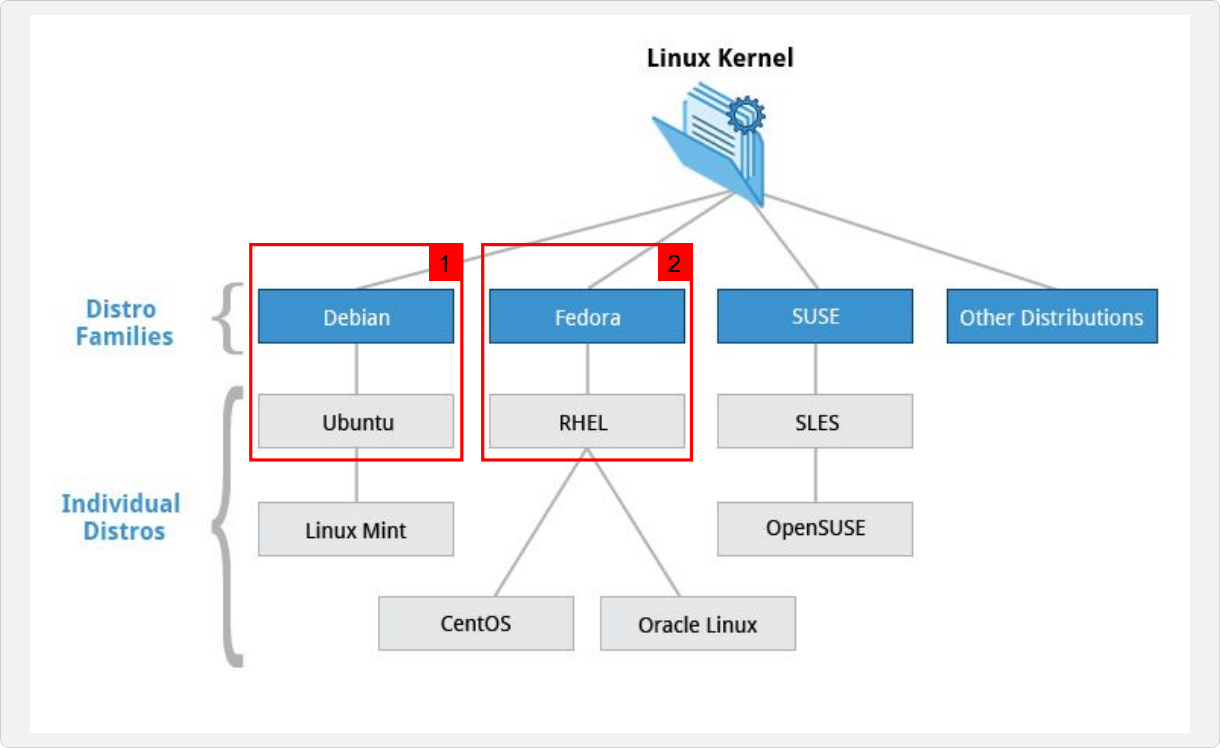


- Introduction to Linux

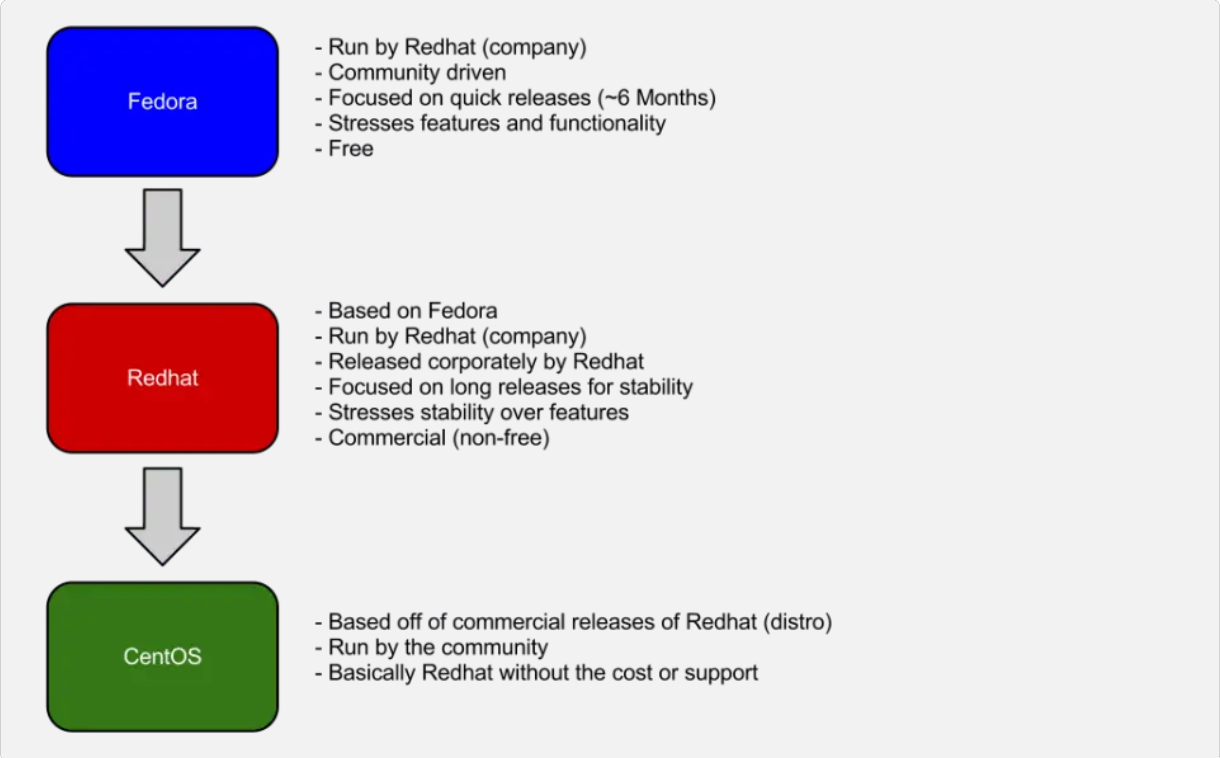
Linux Distribution Families

There are three major distribution families:

- Debian Family Systems (such as Ubuntu)
- Fedora (or Redhat) Family Systems (such as CentOS)
- SUSE Family Systems (such as openSUSE)



- The Difference Between Fedora, Redhat, and CentOS



- Why is Ubuntu Linux the leading choice to replace CentOS for Finserv infrastructure?

Operating systems are the foundation blocks of technology stacks in organisations. When considering an open source operating system for Finserv infrastructure, there are four factors that are key to any enterprise using it – maintainability, continuity, stability and security. The Financial Services industry have started exploring options for a stable and supported open source Linux OS following **IBM Red Hat announcement to accelerate the end-of-life for CentOS 8 with no further operating system updates after December 31, 2021**.

Finservs that have been using CentOS as a stable point distribution for their servers, virtual machines, and appliances had migrated to CentOS 8 expecting support until 2029—only to find out that their “until-2029” distro became “until-2021” distro just a few months after upgrading.

Finservs need a secure, open source Linux distro that can provide long term continuity and maintainability. This blog provides an **overview of why Ubuntu is the leading choice** for a secure, stable platform for finserv infrastructure and cloud native banking.

2.3 Setting up Python Development Environments on Ubuntu

Effective Linux & Bash for Data Scientists

- Linux Distributions / Distros
- Basic Commands
- The Shell
- Text editors in the terminal: nano, vim (or vi)
- Keep calm and **RTFM (Read The F..ing Manual)**

In this section, please open **(Windows) Terminal as Administrator** and run the shell scripts:

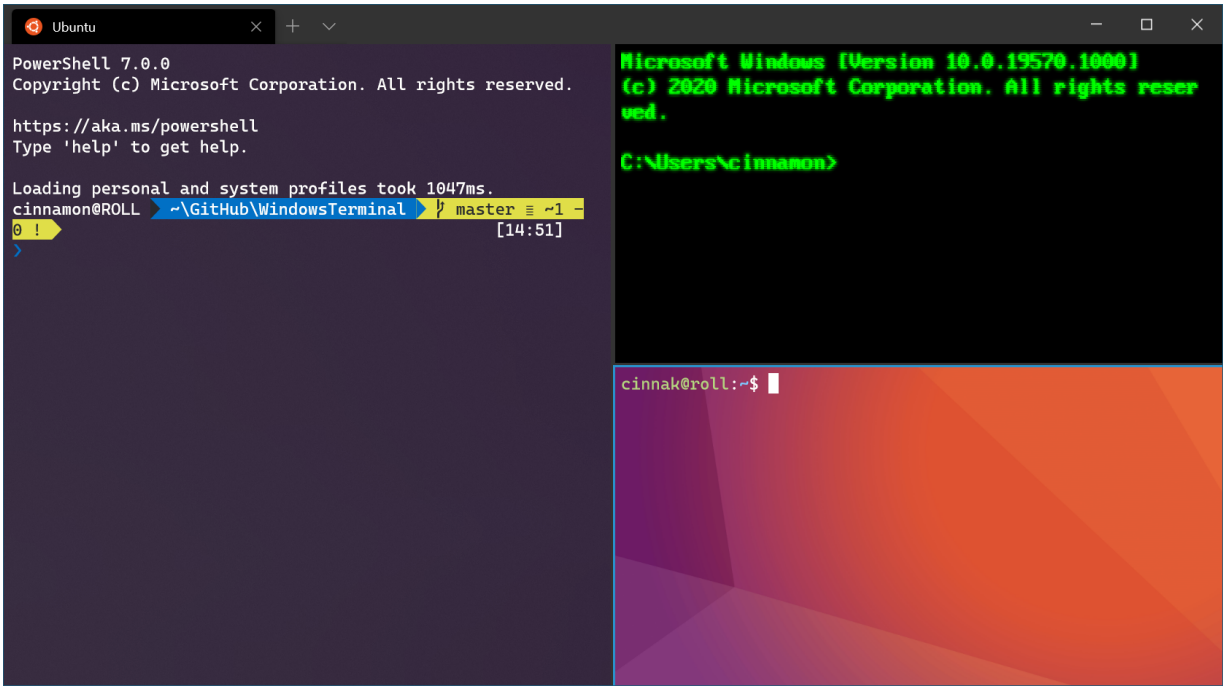
- `2-python-on-linux/setups/linux/ubuntu/_setup-all.sh`

```
#!/usr/bin/env bash

SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && pwd -P)"

. "${SCRIPT_HOME}"/1-sys-deps.sh
. "${SCRIPT_HOME}"/2-py-runtimes.sh
. "${SCRIPT_HOME}"/3-vscode-ext.sh
```

2.3.1 Prerequisites: Windows Terminal Installation and Ubuntu Initialization

Windows Terminal Installation

- What is Windows Terminal?



> **Windows Terminal** is **a modern terminal application** for users of command-line tools and shells like **Command Prompt, PowerShell, and Windows Subsystem for Linux (WSL)**. Its main features include multiple tabs, panes, Unicode and UTF-8 character support, a GPU accelerated text rendering engine, and the ability to create your own themes and customize text, colors, backgrounds, and shortcuts.

- Install and set up Windows Terminal

  Open **Command Prompt** and run:

```
start https://aka.ms/terminal
```

Configuring Ubuntu's Advanced Packaging Tool (APT) Repository for the Fastest Repository Mirror

Setting the Fastest Repository Mirror using Command Line

- Ubuntu Repositories/CommandLine

> This page describes **how to manage software repositories from the command line**. (GUI tools are also available: Managing Repositories in Ubuntu or Kubuntu).
>
> Ubuntu uses **apt** **for package management**. Apt stores a list of repositories or software channels in the file `/etc/apt/sources.list` and in any file with the suffix .list under the directory `/etc/apt/sources.list.d/`

- Updating sources.list:
  `2-python-on-linux/setups/linux/ubuntu/_init/apt-src-update.sh`

```
#!/usr/bin/env bash

# ##############################################################################
# Setting the Fastest Repository Mirror
# ##############################################################################

# Repositories/CommandLine
# https://help.ubuntu.com/community/Repositories/CommandLine

# Official Archive Mirrors for Ubuntu
# https://launchpad.net/ubuntu/+archivemirrors

# Ubuntu mirror "Kakao Corp."
# https://launchpad.net/ubuntu/+mirror/mirror.kakao.com-archive

APT_SRC="/etc/apt/sources.list"

# ==============================================================================
# Backing up the existing sources.list file
# ==============================================================================

sudo cp ${APT_SRC}{,.bak}

# ==============================================================================
# Setting the Fastest Repository Mirror
# ==============================================================================

# ------------------------------------------
# Updating sources.list
# ------------------------------------------

if $(grep -R 'kr.archive.ubuntu.com' ${APT_SRC} > /dev/null) ; then
    sudo sed -i 's/kr.archive.ubuntu.com/mirror.kakao.com/g' ${APT_SRC}
```

```
else
    sudo sed -i 's/archive.ubuntu.com/mirror.kakao.com/g' ${APT_SRC}
fi

sudo sed -i 's/security.ubuntu.com/mirror.kakao.com/g' ${APT_SRC}

sudo apt clean

sudo apt update \
    && sudo apt -y upgrade \
    && sudo apt -y autoremove
```

Setting up Ubuntu packages

- Installing frequently used commands:
  `2-python-on-linux/setups/linux/ubuntu/_init/ubuntu-cmd-utils.sh`

```
#!/usr/bin/env bash

# ================================================================================
# Setting up Ubuntu Packages
# ================================================================================

sudo apt update

sudo apt -y install --no-install-recommends \
    curl \
    wget \
    zip \
    tar \
    sed \
    gawk \
    apt-transport-https \
    ca-certificates \
    net-tools \
    git
```

Configuring Locale: Language, Region, Character Encoding

- Locale Configurations for Korean:
  `2-python-on-linux/setups/linux/ubuntu/_init/locale-setup.sh`

```
#!/usr/bin/env bash

# ================================================================================
# Locale configurations
# ================================================================================

# Locale
# https://help.ubuntu.com/community/Locale

sudo apt -y install locales
sudo apt -y install language-pack-ko

sudo locale-gen ko_KR ko_KR.UTF-8
sudo update-locale LANG=ko_KR.UTF-8 LANGUAGE=ko_KR.UTF-8
```

2.3.2 Installing Python Runtimes and Creating Virtual Environments on Ubuntu

Installing Multiple Python Runtimes

- Installing Multiple Python Runtimes using the deadsnakes PPA (`ppa:deadsnakes/ppa`): `2-python-on-linux/setups/linux/ubuntu/2-py-runtimes.sh`

```
#!/usr/bin/env bash

# ##############################################################################
# Setting up Python Runtime
# ##############################################################################

# https://hackersandslackers.com/multiple-versions-python-ubuntu/
# https://stackoverflow.com/questions/41986507/unable-to-set-default-python-version-to-python3-in-ubuntu

# ==============================================================================
# Prerequisites
# ==============================================================================

# Update and upgrade Ubuntu
sudo apt update \
    && sudo apt -y upgrade \
    && sudo apt -y autoremove

# ------------------------------------------
# compilers and builders
# ------------------------------------------
# https://packages.ubuntu.com/focal/build-essential
# https://packages.ubuntu.com/focal/cmake
# https://packages.ubuntu.com/focal/pkg-config

sudo apt -y install build-essential \
                    cmake \
                    pkg-config

# ------------------------------------------
# PycURL prerequisites
# ------------------------------------------
# https://pypi.org/project/pycurl/
# https://packages.ubuntu.com/focal/libcurl4-openssl-dev
# https://packages.ubuntu.com/focal/libssl-dev

sudo apt -y install libcurl4-openssl-dev \
                    libssl-dev

# ==============================================================================
# Setting up Python Interpreters
# ==============================================================================

# ------------------------------------------
# Checking the latest Python Interpreter
# ------------------------------------------

apt-cache search ^python3

# ------------------------------------------
# Adding the deadsnakes PPA
# ------------------------------------------

sudo add-apt-repository -y ppa:deadsnakes/ppa
sudo apt update
```

```
56
57  # ----------------------------------------
58  # Installing Python Interpreters on Ubuntu with Apt
59  # ----------------------------------------
60  # Ubuntu 18.04 (bionic) installed Python version: 3.6
61  # Ubuntu 20.04 (focal) installed Python version: 3.8
62
63  # https://packages.ubuntu.com/bionic-updates/python3.8-dev
64  # https://packages.ubuntu.com/focal/python3.8-dev
65
66  python_versions=(
67      'python3.6'
68      'python3.7'
69      'python3.8'
70      'python3.9'
71  )
72
73  for py_ver in ${python_versions[@]}; do
74      sudo apt -y install ${py_ver}-dev
75  done
76
77  # ----------------------------------------
78  # Configuring System Python Interpreter
79  # ----------------------------------------
80
81  # update-alternatives
82  # http://manpages.ubuntu.com/manpages/trusty/man8/update-alternatives.8.html
83
84  # checking installed python versions
85  ls -al /usr/bin/python*
86
87  # checking available python versions
88  # NOTICE: update-alternatives: error: no alternatives for python3
89  update-alternatives --list python3
90  update-alternatives --display python3
91
92  # setting python3 and python alternatives with priority numbers
93  i=0
94  for py_ver in ${python_versions[@]}; do
95      ((i++))
96      sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/${py_ver} $
        {i}
97      sudo update-alternatives --install /usr/bin/python python /usr/bin/${py_ver} ${i
        }
98  done
99
100 # selecting default python3
101 sudo update-alternatives --config python3
102
103 # selecting default python3
104 sudo update-alternatives --config python
105
106 # chekcing python versions
107 python3 -V
108 # python2 -V
109 python -V
110
111 # removing python alternatives
112 # sudo update-alternatives --remove python3 /usr/bin/python3.6
113
114 # ================================================================================
115 # Setting up pip for newly installed Python
116 # ================================================================================
```

```
117
118 # ---------------------------------------
119 # Checking the latest pip version
120 # ---------------------------------------
121
122 # apt-cache search *-pip$
123
124 # ---------------------------------------
125 # Installing pip on Ubuntu with Apt
126 # ---------------------------------------
127
128 # installing pip for 3.6 first
129 # https://packages.ubuntu.com/focal/python3-pip
130 sudo apt -y install python3-pip
131
132 python -m pip install --upgrade pip
133
134 # ---------------------------------------
135 # Configuring pip
136 # ---------------------------------------
137
138 # selecting default pip
139 sudo update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 1
140
141 # ---------------------------------------
142 # Miscellaneous Troubleshooting
143 # ---------------------------------------
144
145 # How to fix 'ModuleNotFoundError: No module named 'apt_pkg'?
146 # https://stackoverflow.com/questions/56218562/how-to-fix-modulenotfounderror-no-
    module-named-apt-pkg
147 sudo apt -y remove python3-apt
148 sudo apt -y install python3-apt
149
150 # Installing built-package format for Python
151 # https://stackoverflow.com/questions/34819221/why-is-python-setup-py-saying-invalid
    -command-bdist-wheel-on-travis-ci
152 # https://packages.ubuntu.com/focal/python3-wheel
153 sudo apt -y install python3-wheel
154
155 # PyUnit extension for managing expensive test fixtures - Python 3.x
156 # https://packages.ubuntu.com/focal/python3-testresources
157 sudo apt -y install python3-testresources
158
159 # ============================================================================
160 # Setting up package for virtual environment
161 # ============================================================================
162
163 # python -m pip install virtualenv
164 # https://packages.ubuntu.com/focal/python3.8-venv
165 # https://packages.ubuntu.com/focal/python3.9-venv
166 sudo apt -y install python3.6-venv
167 sudo apt -y install python3.7-venv
168 sudo apt -y install python3.8-venv
169 sudo apt -y install python3.9-venv
```

Creating Python Virtual Environments and Installing Python Packages

(1) Selecting Python interpreter version

```
sudo update-alternatives --config python3
sudo update-alternatives --config python
```

(2) Creating a virtual environment **in a Python project**

```
python -m venv .venv
```

(3) **Activating the virtual environment**

```
. .venv/bin/activate
```

(4) Installing Python packages

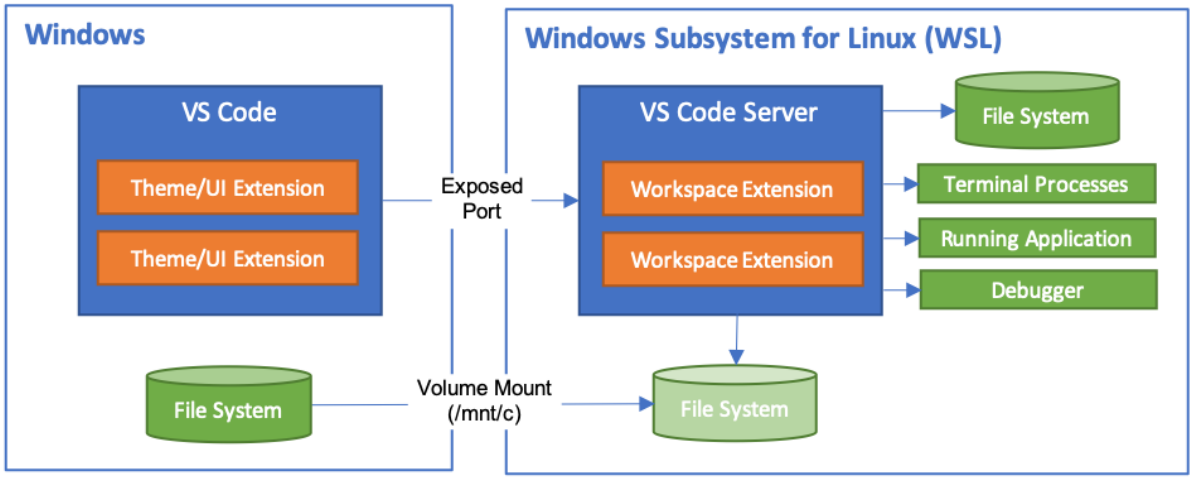
```
pip install -r requirements/dev.txt
```

2.4 Programming Python in VS Code on WSL

2.4.1 Understanding VS Code Interaction between Windows and WSL

WSL Architecture for VS Code

- Developing in WSL



> The **Visual Studio Code Remote - WSL** extension lets you use the **Windows Subsystem for Linux (WSL)** as your full-time development environment right from VS Code. You can develop in a Linux-based environment, use Linux-specific toolchains and utilities, and run and debug your Linux-based applications all from the comfort of Windows.
>
> The extension runs commands and other extensions directly in WSL so you can edit files located in WSL or the mounted Windows filesystem (for example `/mnt/c`) without worrying about pathing issues, binary compatibility, or other cross-OS challenges.
>
> This lets VS Code provide a **local-quality development experience** — including full IntelliSense (completions), code navigation, and debugging — **regardless of where your code is hosted**.

- workspace path on **Windows**:
  `C:\Users\siai\workspaces\issues-in-computer-programming\`
- workspace path on **Ubuntu**:
  `/mnt/c/Users/siai/workspaces/issues-in-computer-programming/`

  NOTICE: Under `/mnt/*` folders, WSL has some performance issues, so that it is recommended to use folders under **/home/*** other than `/mnt/*` folders.

File systems in WSL for Performance Issues

- DrvFs

> To facilitate interoperability with Windows, WSL uses the DrvFs file system. **WSL automatically mounts all fixed drives with supported file systems under `/mnt`, such as `/mnt/c`, `/mnt/d`, etc.** Currently, only NTFS and ReFS volumes are supported.

- Major performance (I/O?) issue in **/mnt/*** and in **(home)**

> I'm having a major performance issue when using **`/mnt/*` folders**. If I set up a Symfony project under /mnt/c, it is really slow. If I set it up **under `/home/` mikael**, it is very fast.

2.4.2 Installing VS Code Extensions on Windows and Ubuntu

Installing VS Code Extensions on Windows

- Remote - WSL

> The **Remote - WSL extension** lets you use VS Code on Windows to build Linux applications that run on the **Windows Subsystem for Linux (WSL)**. You get all the productivity of Windows while developing with Linux-based tools, runtimes, and utilities.

Installing VS Code Extensions on Ubuntu

- Python extension for Visual Studio Code

> A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: >=3.6), including features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more!
>
> **The Python extension will automatically install the Pylance and Jupyter extensions** to give you the best experience when working with Python files and Jupyter notebooks. However, Pylance is an optional dependency, meaning the Python extension will remain fully functional if it fails to be installed. You can also uninstall it at the expense of some features if you're using a different language server.

2.4.3 Running Python Scripts in VS Code on Ubuntu

Opening a remote folder or workspace

Open a remote folder or workspace

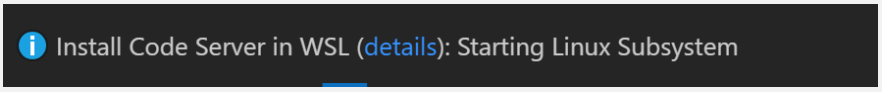
- From VS Code

> You can open a Remote WSL window directly from VS Code:
>
> 1. Start VS Code.
> 2. Press `F1`, select **`Remote-WSL: New Window`** for the default distro or **`Remote-WSL: New Window using Distro`** for a specific distro.
> 3. Use the File menu to open your folder.

- From the WSL terminal

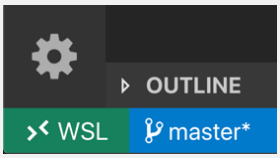
> Opening a folder inside the Windows Subsystem for Linux in VS Code is very similar to opening up a Windows folder from the command prompt or PowerShell.
>
> 1. Open a **WSL terminal window** (using the start menu item or by typing `wsl` from a command prompt / PowerShell).
> 2. Navigate to a folder you'd like to open in VS Code (including, but not limited to, Windows filesystem mounts like `/mnt/c`)
> 3. Type **`code .`** in the terminal. When doing this for the first time, you should see VS Code fetching components needed to run in WSL. This should only take a short while, and is only needed once.
>
>   Note: If this command does not work, you may need to restart your terminal or you may not have added VS Code to your path when it was installed.

4. After a moment, a new VS Code window will appear, and you'll see a notification that VS Code is opening the folder in WSL.



VS Code will now continue to configure itself in WSL and keep you up to date as it makes progress.

5. Once finished, you now see a WSL indicator in the bottom left corner, and you'll be able to use VS Code as you would normally!



Running Python Scripts with Hello World Example in VS Code

Please refer the `1.1.1 Running Python Scripts with Hello World Example in VS Code`:

- (3) Selecting a Python Interpreter: `.venv/bin/python`
- (4) Creating and editing a Python Script file: `src/hello.py`
- (5) Running Python Scripts in VS Code

2.5 Automation for Programming Python on Linux

2.5.1 Batch Scripts to only Install once Required Software on Windows

`2-python-on-linux/setups/windows/`
Windows commands

(1) `prerequisites.bat`

- Installing Prerequisites: Chocolatey and System Utilities

```
@echo off

pushd "%~dp0"

echo ############################################################################
echo Install Prerequisites
echo ############################################################################

set SCRIPT_HOME=%cd%

call "%SCRIPT_HOME%\prereqs\win-pkg-mgr.bat"

call "%SCRIPT_HOME%\prereqs\system-prereqs.bat"
```

- Installing Windows Package Manager: Chocolatey
  INSTALLING CHOCOLATEY, Search Chocolatey Packages

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
    InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.
    Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET
    "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

- Installing GNU related and other utilities for Windows

```
@echo off

echo ############################################################################
echo Installing GNU related and other utilities for Windows
echo ############################################################################

echo ============================================================================
echo File Utilities
echo ============================================================================

echo ----------------------------------------------------------------------------
echo Get Files
echo ----------------------------------------------------------------------------

:: GNU Wget
:: https://chocolatey.org/packages/Wget
choco install -y wget --version 1.20

:: cURL
:: https://chocolatey.org/packages/curl
choco install -y curl

:: WinSCP (Install)
:: https://chocolatey.org/packages/winscp.install
choco install -y winscp.install

echo ----------------------------------------------------------------------------
echo Compress and Extract Files
```

```
29 echo ------------------------------------------------------------------------
30
31 :: 7-Zip (Install)
32 :: https://chocolatey.org/packages/7zip.install
33 choco install -y 7zip.install
34 :: choco upgrade -y 7zip.install
35
36 :: Info-ZIP
37 :: https://chocolatey.org/packages/unzip
38 choco install -y unzip
39
40 :: TarTool
41 :: https://chocolatey.org/packages/tartool
42 choco install -y tartool
43
44 echo ------------------------------------------------------------------------
45 echo Find and Replace Text in Files
46 echo ------------------------------------------------------------------------
47
48 :: GNU grep
49 :: https://chocolatey.org/packages/grep
50 :: choco install -y grep
51
52 :: grepWin
53 :: https://chocolatey.org/packages/grepwin
54 :: choco install -y grepwin
55
56 :: GNU sed
57 :: https://chocolatey.org/packages/sed
58 choco install -y sed
59
60 echo ========================================================================
61 echo JSON Utilities
62 echo ========================================================================
63
64 :: jq
65 :: https://chocolatey.org/packages/jq
66 choco install -y jq
67
68 echo ========================================================================
69 echo Miscellaneous
70 echo ========================================================================
71
72 :: GNU Make
73 choco install -y make
```

(2) `hyper-v/enable-hyper-v.bat`

- Enabling Hyper-V
  Install Hyper-V on Windows 10

```
1  @echo off
2
3  :: Install Hyper-V on Windows 10
4  :: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-
       start/enable-hyper-v
5
6  echo ========================================================================
7  echo Enabling Hyper-V with CMD and DISM
8  echo ========================================================================
9
10 :: powershell Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper
       -V -All
```

```
11 powershell dism.exe /online /enable-feature /featurename:Microsoft-Hyper-V /all
       /norestart
```

(3) `wsl/wsl-instl-1.bat`

- Installing Windows Subsystem for Linux (WSL): Step 1 to 3
  Windows Subsystem for Linux Installation Guide for Windows 10

```
1  @echo off
2
3  :: Windows Subsystem for Linux Installation Guide for Windows 10
4  :: https://docs.microsoft.com/en-us/windows/wsl/install-win10
5
6  echo ########################################################################
7  echo Windows Subsystem for Linux Installation Guide for Windows 10
8  echo ########################################################################
9
10 echo ========================================================================
11 echo Install Windows Subsystem for Linux
12 echo ========================================================================
13
14 echo ------------------------------------------------------------------------
15 echo Step 1 - Enable the Windows Subsystem for Linux
16 echo ------------------------------------------------------------------------
17
18 :: Open PowerShell as Administrator and run below
19 powershell dism.exe /online /enable-feature /featurename:Microsoft-Windows-
       Subsystem-Linux /all /norestart
20
21 echo ------------------------------------------------------------------------
22 echo Step 2 - Check requirements for running WSL 2
23 echo ------------------------------------------------------------------------
24
25 :: Check your Windows version (2004, Build 19041 or higher) by selecting the
       Windows logo key + R, type winver, select OK
26 winver
27
28 echo ------------------------------------------------------------------------
29 echo Step 3 - Enable Virtual Machine feature
30 echo ------------------------------------------------------------------------
31
32 :: Open PowerShell as Administrator and run below
33 powershell dism.exe /online /enable-feature /featurename:VirtualMachinePlatform
       /all /norestart
34
35 :: Restart your machine to complete the WSL install and update to WSL 2
36 shutdown /r /t 10
```

(4) `wsl/wsl-instl-2.bat`

- Installing Windows Subsystem for Linux (WSL): Step 4 to 6
  Windows Subsystem for Linux Installation Guide for Windows 10

```
1  @echo off
2
3  :: Windows Subsystem for Linux Installation Guide for Windows 10
4  :: https://docs.microsoft.com/en-us/windows/wsl/install-win10
5
6  echo ########################################################################
7  echo Windows Subsystem for Linux Installation Guide for Windows 10
8  echo ########################################################################
9
```

```
echo ========================================================================
echo Install Windows Subsystem for Linux
echo ========================================================================

echo ------------------------------------------------------------------------
echo Step 4 - Download the Linux kernel update package
echo ------------------------------------------------------------------------

:: TODO: update WSL 2 Linux kernel
:: https://aka.ms/wsl2kernel

curl https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi -o
    %USERPROFILE%\Downloads\wsl_update_x64.msi
%USERPROFILE%\Downloads\wsl_update_x64.msi

echo ------------------------------------------------------------------------
echo Step 5 - Set WSL 2 as your default version
echo ------------------------------------------------------------------------

wsl --set-default-version 2

:: Set a distro to be backed by WSL 2
:: wsl --set-version Ubuntu-20.04 2

:: Verifying what versions of WSL your distro are using
:: wsl --list --verbose
:: wsl -l -v

echo ------------------------------------------------------------------------
echo Step 6 - Install your Linux distribution of choice
echo ------------------------------------------------------------------------

:: Microsoft Store > Search > "Ubuntu LTS" > Ubuntu 18.04 LTS
:: start https://www.microsoft.com/store/apps/9N9TNGVNDL3Q

:: TODO: install Ubuntu 20.04+ LTS using Microsoft Store
start https://www.microsoft.com/store/apps/9n6svws3rx71
```

2.5.2 Shell Scripts to only Install once Required Software on Ubuntu

`2-python-on-linux/setups/linux/ubuntu`
The Linux command line for beginners

(1) `1-sys-deps.sh`

- Installing Initial Ubuntu Packages

```
#!/usr/bin/env bash

if [[ -z "${SCRIPT_HOME}" ]]; then
    SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && pwd -P)"
fi

. "${SCRIPT_HOME}"/_init/_init-setup.sh
```

- `_init-setup.sh`

```
# Checking the variable, Init Home Path
if [[ -z "${INIT_HOME}" ]]; then
    INIT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && pwd -P)"
fi

```

```
12 # Setting the Fastest Repository Mirror
13 . "${INIT_HOME}"/apt-src-update.sh
14
15 # Setting up Ubuntu Packages
16 . "${INIT_HOME}"/ubuntu-cmd-utils.sh
17
18 # Locale configurations
19 . "${INIT_HOME}"/locale-setup.sh
```

- `apt-src-update.sh`

```
16 APT_SRC="/etc/apt/sources.list"
17
18 # ==========================================================================
19 # Backing up the existing sources.list file
20 # ==========================================================================
21
22 sudo cp ${APT_SRC}{,.bak}
23
24 # ==========================================================================
25 # Setting the Fastest Repository Mirror
26 # ==========================================================================
27
28 # ----------------------------------------
29 # Updating sources.list
30 # ----------------------------------------
31
32 if $(grep -R 'kr.archive.ubuntu.com' ${APT_SRC} > /dev/null) ; then
33     sudo sed -i 's/kr.archive.ubuntu.com/mirror.kakao.com/g' ${APT_SRC}
34 else
35     sudo sed -i 's/archive.ubuntu.com/mirror.kakao.com/g' ${APT_SRC}
36 fi
37
38 sudo sed -i 's/security.ubuntu.com/mirror.kakao.com/g' ${APT_SRC}
39
40 sudo apt clean
41
42 sudo apt update \
43     && sudo apt -y upgrade \
44     && sudo apt -y autoremove
```

- `ubuntu-cmd-utils.sh`

```
7  sudo apt update
8
9  sudo apt -y install --no-install-recommends \
10     curl \
11     wget \
12     zip \
13     tar \
14     sed \
15     gawk \
16     apt-transport-https \
17     ca-certificates \
18     net-tools \
19     git
```

- `locale-setup.sh`

```
10 sudo apt -y install locales
11 sudo apt -y install language-pack-ko
12
13 sudo locale-gen ko_KR ko_KR.UTF-8
14 sudo update-locale LANG=ko_KR.UTF-8 LANGUAGE=ko_KR.UTF-8
```

(2) `2-py-runtimes.sh`

- Installing Python Runtime on Ubuntu using the deadsnakes PPA

```
#!/usr/bin/env bash

# ##############################################################################
# Setting up Python Runtime
# ##############################################################################

# https://hackersandslackers.com/multiple-versions-python-ubuntu/
# https://stackoverflow.com/questions/41986507/unable-to-set-default-python-version-to-python3-in-ubuntu

# ==============================================================================
# Prerequisites
# ==============================================================================

# Update and upgrade Ubuntu
sudo apt update \
    && sudo apt -y upgrade \
    && sudo apt -y autoremove

# ------------------------------------------
# compilers and builders
# ------------------------------------------
# https://packages.ubuntu.com/focal/build-essential
# https://packages.ubuntu.com/focal/cmake
# https://packages.ubuntu.com/focal/pkg-config

sudo apt -y install build-essential \
                    cmake \
                    pkg-config

# ------------------------------------------
# PycURL prerequisites
# ------------------------------------------
# https://pypi.org/project/pycurl/
# https://packages.ubuntu.com/focal/libcurl4-openssl-dev
# https://packages.ubuntu.com/focal/libssl-dev

sudo apt -y install libcurl4-openssl-dev \
                    libssl-dev

# ==============================================================================
# Setting up Python Interpreters
# ==============================================================================

# ------------------------------------------
# Checking the latest Python Interpreter
# ------------------------------------------

apt-cache search ^python3

# ------------------------------------------
# Adding the deadsnakes PPA
# ------------------------------------------

sudo add-apt-repository -y ppa:deadsnakes/ppa
sudo apt update

# ------------------------------------------
# Installing Python Interpreters on Ubuntu with Apt
# ------------------------------------------
```

```
60  # Ubuntu 18.04 (bionic) installed Python version: 3.6
61  # Ubuntu 20.04 (focal) installed Python version: 3.8
62
63  # https://packages.ubuntu.com/bionic-updates/python3.8-dev
64  # https://packages.ubuntu.com/focal/python3.8-dev
65
66  python_versions=(
67      'python3.6'
68      'python3.7'
69      'python3.8'
70      'python3.9'
71  )
72
73  for py_ver in ${python_versions[@]}; do
74      sudo apt -y install ${py_ver}-dev
75  done
76
77  # ----------------------------------------
78  # Configuring System Python Interpreter
79  # ----------------------------------------
80
81  # update-alternatives
82  # http://manpages.ubuntu.com/manpages/trusty/man8/update-alternatives.8.html
83
84  # checking installed python versions
85  ls -al /usr/bin/python*
86
87  # checking available python versions
88  # NOTICE: update-alternatives: error: no alternatives for python3
89  update-alternatives --list python3
90  update-alternatives --display python3
91
92  # setting python3 and python alternatives with priority numbers
93  i=0
94  for py_ver in ${python_versions[@]}; do
95      ((i++))
96      sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/${
        py_ver} ${i}
97      sudo update-alternatives --install /usr/bin/python python /usr/bin/${py_ver}
         ${i}
98  done
99
100 # selecting default python3
101 sudo update-alternatives --config python3
102
103 # selecting default python3
104 sudo update-alternatives --config python
105
106 # chekcing python versions
107 python3 -V
108 # python2 -V
109 python -V
110
111 # removing python alternatives
112 # sudo update-alternatives --remove python3 /usr/bin/python3.6
113
114 # ================================================================================
115 # Setting up pip for newly installed Python
116 # ================================================================================
117
118 # ----------------------------------------
119 # Checking the latest pip version
120 # ----------------------------------------
```

```
121
122 # apt-cache search *-pip$
123
124 # ----------------------------------------
125 # Installing pip on Ubuntu with Apt
126 # ----------------------------------------
127
128 # installing pip for 3.6 first
129 # https://packages.ubuntu.com/focal/python3-pip
130 sudo apt -y install python3-pip
131
132 python -m pip install --upgrade pip
133
134 # ----------------------------------------
135 # Configuring pip
136 # ----------------------------------------
137
138 # selecting default pip
139 sudo update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 1
140
141 # ----------------------------------------
142 # Miscellaneous Troubleshooting
143 # ----------------------------------------
144
145 # How to fix 'ModuleNotFoundError: No module named 'apt_pkg'?
146 # https://stackoverflow.com/questions/56218562/how-to-fix-modulenotfounderror-no
    -module-named-apt-pkg
147 sudo apt -y remove python3-apt
148 sudo apt -y install python3-apt
149
150 # Installing built-package format for Python
151 # https://stackoverflow.com/questions/34819221/why-is-python-setup-py-saying-
    invalid-command-bdist-wheel-on-travis-ci
152 # https://packages.ubuntu.com/focal/python3-wheel
153 sudo apt -y install python3-wheel
154
155 # PyUnit extension for managing expensive test fixtures - Python 3.x
156 # https://packages.ubuntu.com/focal/python3-testresources
157 sudo apt -y install python3-testresources
158
159 # ============================================================================
160 # Setting up package for virtual environment
161 # ============================================================================
162
163 # python -m pip install virtualenv
164 # https://packages.ubuntu.com/focal/python3.8-venv
165 # https://packages.ubuntu.com/focal/python3.9-venv
166 sudo apt -y install python3.6-venv
167 sudo apt -y install python3.7-venv
168 sudo apt -y install python3.8-venv
169 sudo apt -y install python3.9-venv
```

(3) `3-vs-code-ext.sh`

- Installing VS Code Extensions for Python Development
  Command line extension management

```
3 if [[ -z "${SCRIPT_HOME}" ]]; then
4     SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && pwd -P)"
5 fi
6
7 . "${SCRIPT_HOME}"/vscode/vscode-ext-py.sh
```

- `vscode-ext-py.sh`

```
#!/usr/bin/env bash

# ##############################################################################
# Installing Extensions of Visual Studio Code (VS Code) for Python
# ##############################################################################

# Python in Visual Studio Code
# https://code.visualstudio.com/docs/languages/python

# Python Development in Visual Studio Code
# https://realpython.com/python-development-visual-studio-code/

if [[ -z "${SCRIPT_HOME}" ]]; then
    SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && cd ../ && pwd -P)"
fi

# ==============================================================================
# Prerequisites: code
# ==============================================================================

. "${SCRIPT_HOME}"/cmd/cmd-exists.sh 'code'

# ==============================================================================
# Installing VS Code Extensions for Python Development
# ==============================================================================

# Pylance
# A performant, feature-rich language server for Python in VS Code
# https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance
# code --install-extension ms-python.vscode-pylance

# Python Extension Pack
# Popular Visual Studio Code extensions for Python
# https://marketplace.visualstudio.com/items?itemName=donjayamanne.python-extension-pack
#   * Python
#     Linting, Debugging (multi-threaded, remote), Intellisense, Jupyter Notebooks, code formatting, refactoring, unit tests, and more.
#     https://marketplace.visualstudio.com/items?itemName=ms-python.python
#   * MagicPython
#     Syntax highlighter for cutting edge Python.
#     https://marketplace.visualstudio.com/items?itemName=magicstack.MagicPython
#   * Jinja
#     Jinja template language support for Visual Studio Code
#     https://marketplace.visualstudio.com/items?itemName=wholroyd.jinja
#   * Django
#     Beautiful syntax and scoped snippets for perfectionists with deadlines
#     https://marketplace.visualstudio.com/items?itemName=batisteo.vscode-django
#   * Visual Studio IntelliCode
#     AI-assisted development
#     https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.vscodeintellicode
code --install-extension donjayamanne.python-extension-pack

# Python Extended (Snippets)
# Python Extended is a vscode snippet that makes it easy to write codes in python by providing completion options along with all arguments.
# https://marketplace.visualstudio.com/items?itemName=tushortz.python-extended-snippets
# code --install-extension tushortz.python-extended-snippets@0.0.1
# code --install-extension tushortz.python-extended-snippets

```

```
58 # Python Indent
59 # Correct python indentation.
60 # https://marketplace.visualstudio.com/items?itemName=KevinRose.vsc-python-
       indent
61 code --install-extension KevinRose.vsc-python-indent
62
63 # AREPL for python
64 # real-time python scratchpad
65 # https://marketplace.visualstudio.com/items?itemName=almenon.arepl
66 code --install-extension almenon.arepl
67
68 # Python Docstring Generator
69 # Automatically generates detailed docstrings for python functions
70 # https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring
71 code --install-extension njpwerner.autodocstring
```

(4) `_setup-all.sh`

- Installing all required software with just one shell script file

```
1 #!/usr/bin/env bash
2
3 SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && pwd -P)"
4
5 . "${SCRIPT_HOME}"/1-sys-deps.sh
6 . "${SCRIPT_HOME}"/2-py-runtimes.sh
7 . "${SCRIPT_HOME}"/3-vscode-ext.sh
```

2.5.3 Makefile for Building a Python Project repeatedly

`2-python-on-linux/Makefile`

- GNU Make

(1) **Selecting Python version**

- `make select-py`

```
34 select-py:
35   sudo update-alternatives --config python
```

- `make select-py3`

```
37 select-py3:
38   sudo update-alternatives --config python3
```

(2) Creating a virtual environment

- `make venv`

```
40 venv:
41   python -m venv $(VENV)
```

(3) Activating the virtual environment

- `$(VENV_ACT)`

```
11 PY_VER = $(shell python -V | awk '{print $$2}' | cut -c1-3)
12 VENV = .venv-py$(PY_VER)
13 VENV_ACT = . $(VENV)/bin/activate
```

(4) Installing Python packages

- `make deps`

```
43 deps:
44   $(VENV_ACT) \
45     && pip install -r requirements/prereqs.txt \
46     && pip install -r requirements/dev.txt \
47     && pip list \
48     && pipdeptree \
49     && pipdeptree --json > deps.json \
50     && pipdeptree > deps.txt
```

(5) Creating `.env` file if not exists

- `make .env`

```
52 .env:
53   [ -f .env ] || echo "PYTHONPATH=./src"> .env
```

(6) Running Python scripts

- `make run`

```
55 run:
56   $(VENV_ACT) \
57     && python src/iris_data_analysis.py \
58     && python src/iris_kmeans_dash.py
```

(7) Testing Python scripts

- `make test`

```
87 test:
88   $(VENV_ACT) \
89     && pytest tests/test_settings.py \
90     && pytest tests/test_config_settings.py
```

(8) Generating documentation files

- `make docs`

```
65 docs:
66   make docs-apis
67
68 docs-apis:
69   $(VENV_ACT) \
70   && lazydocs \
71     --output-path="docs/4-apis/md" \
72     --overview-file="index.md" \
73     src \
74   && pdoc \
75     --force \
76     --html \
77     --output-dir docs/4-apis/html \
78     src
```

(9) Removing the virtual environment and temporary folders

- `make clean`

```
clean:
  rm -rf $(VENV)/
  find . -type f -name "*.py[co]" -delete
  find . -type d -name "__pycache__" -delete
  find . -type d -name ".pytest_cache" -exec rm -rf {} \;
  find . -type d -name ".ipynb_checkpoints" -exec rm -rf {} \;
```

(10) Executing all the tasks

- `make all`

```
TARGETS := clean venv deps run test docs
.PHONY: all select-py select-py3 .env $(TARGETS) docs-apis

all:
  make $(TARGETS)
```

2.5.4 Automation in Source Code Level

- Data Science Pipeline Automation with sklearn.pipeline.Pipeline:
  `2-python-on-linux/src/iris_data_analysis.py`

```
—
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=0)

# ----------------------------------------
# Creating Pipelines, Model Fitting and Predicting
# ----------------------------------------

pipeline_lr = Pipeline(
    [
        ("std_scaler1", StandardScaler()),
        ("lr_classifier", LogisticRegression(random_state=0)),
    ]
)

pipeline_dt = Pipeline(
    [("std_scaler2", StandardScaler()), ("dt_classifier", DecisionTreeClassifier())]
)

pipeline_svm = Pipeline(
    [("std_scaler3", StandardScaler()), ("svm_classifier", svm.SVC())]
)

pipelines = [pipeline_lr, pipeline_dt, pipeline_svm]
pipe_dict = {0: "Logistic Regression", 1: "Decision Tree", 2: "Support Vector
    Machine"}

for pipe in pipelines:
    pipe.fit(X_train, y_train)

for i, model in enumerate(pipelines):
    print(f"{pipe_dict[i]} Test Accuracy: {model.score(X_test, y_test)}")

# ----------------------------------------
# Saving and Loading Models
# ----------------------------------------

MODEL_FILE = f"{MODEL_ROOT}/pipelines_models.pkl"

```

```
618 with open(MODEL_FILE, "wb") as file:
619     for pipe in pipelines:
620         pickle.dump(pipe, file)
621 #     pickle.dump(pipeline_lr, file)
622 
623 pipelines2 = []
624 with open(MODEL_FILE, "rb") as file:
625     while True:
626         try:
627             pipelines2.append(pickle.load(file))
628         except EOFError:
629             break
630 
631 for i, model in enumerate(pipelines2):
632     print(i, ":", model)
633     print(f"{pipe_dict[i]} Test Accuracy: {model.score(X_test, y_test)}")
```

Chapter 3

Tracking Changes in a Project and Sharing the Project with Git

3.1 Why Git for Data Science?

Sharing source code and projects

3.1.1 Sharing a (Data Science) Project for Collaboration

(File-based Approach) Sharing a single Python project with multiple writers

(1) Sharing just a Jupyter notebook or one Python script with one writer

- How to check the changes of source code comparing to previous source code?
  - simple source code comparison
  - comments for changes
- How to merge other's source code with your code?
  - other's changes from previous source code
  - my changes from previous source code
  - merging other's and my changes into source code

(2) How to do when expanding to multiple files and multiple writers

3.1.2 Version Control Systems (VCSs)

VCSs for recording changes to files and sharing source code and projects

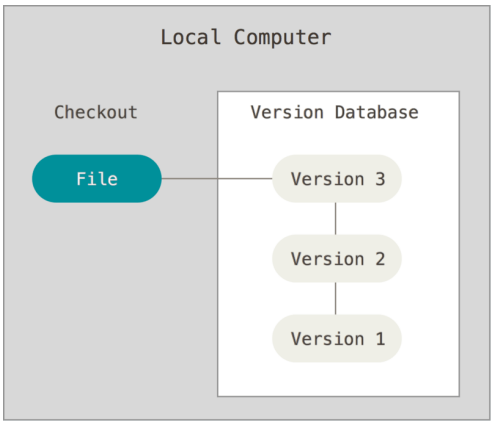
Version Control Systems and Their Generations

About Version Control[1]

> What is "version control", and why should you care? **Version control is a system that records changes to a file or set of files over time** so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.

(1) Local Version Control Systems

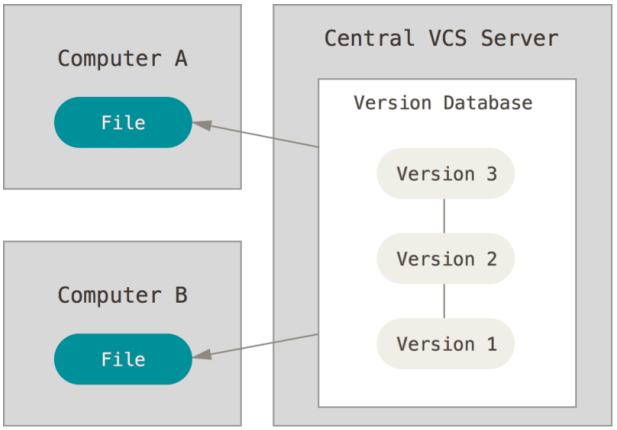
[1] S. Chacon. *Pro Git*. Expert's voice in software development. Apress, 2009. ISBN: 9781430218340. URL: https://git-scm.com/book/en/v2.



Many people's version-control method of choice is to **copy files into another directory** (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

(2) Centralized Version Control Systems

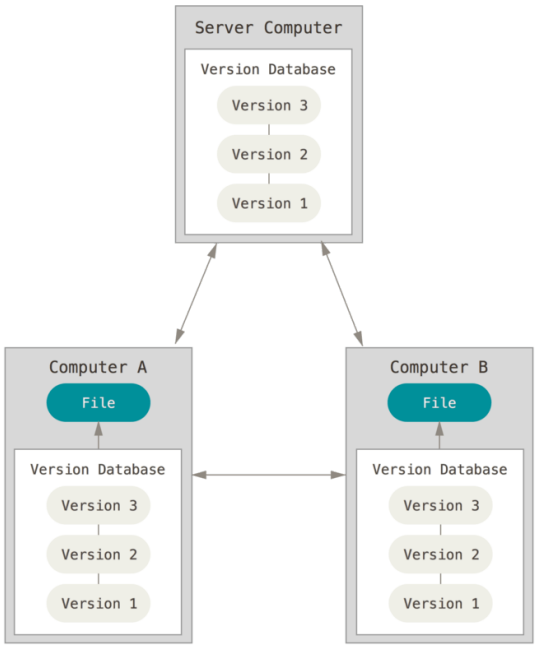


The next major issue that people encounter is that they **need to collaborate with developers on other systems**. To deal with this problem, **Centralized Version Control Systems (CVCSs)** were developed. These systems (such as CVS, **Subversion**, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

This setup offers **many advantages**, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client.

However, this setup also has **some serious downsides**. The most obvious is **the single point of failure** that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on. If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCSs suffer from this same problem — whenever you have the entire history of the project in a single place, you risk losing everything.

(3) Distributed Version Control Systems



This is where **Distributed Version Control Systems (DVCSs)** step in. In a DVCS (such as **Git**, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, **they fully mirror the repository, including its full history**. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

3.1.3 Why Git?

- Companies & Projects Using Git

- What are some use cases in big projects where Git performs better than SVN?

A Distributed version control system

- Clones a project you not only get the files but the **complete history of that project** along with it.
- To do a large commit in SVN, it's painful and highly unproductive.
- **Eliminates any single point of failure** which SVN is prone too.

Branching and Merging

- In Subversion, the process is extremely messy and prone to conflicts, not to mention in Subversion operations like branching and merging are extremely slow.

- A Short History of Git

As with many great things in life, Git began with a bit of creative destruction and fiery controversy.

The Linux kernel is an open source software project of fairly large scope. For most of the lifetime of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files. In 2002, **the Linux kernel project** began using a proprietary **DVCS called BitKeeper**.

In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the **tool's free-of-charge status was revoked**. This prompted **the Linux development community (and in particular Linus Torvalds, the creator of Linux)** to develop **their own tool** based on some of the lessons they learned while using BitKeeper. Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (**thousands** of parallel **branches**)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities. It's amazingly fast, it's very efficient with large projects, and it has an incredible branching system for non-linear development (See Git Branching).

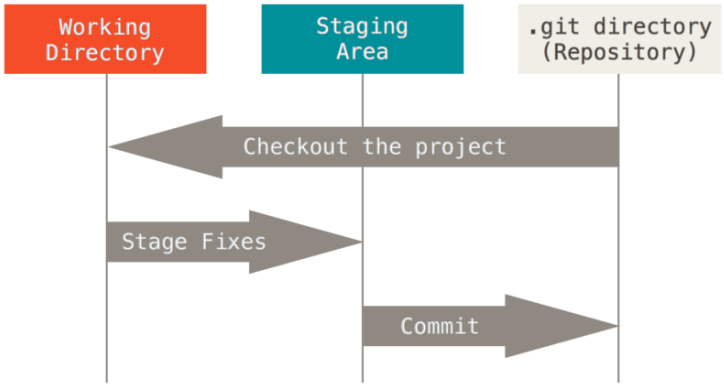
- Git Internals - Git Objects

Git is a content-addressable filesystem. Great. What does that mean? It means that at the core of Git is a **simple key-value data store**. What this means is that you **can insert any kind of content into a Git repository**, for which Git will **hand you back a unique key** you can use later to retrieve that content.

3.2 The Key Concepts for Using Git

3.2.1 The Three States

- The Three States

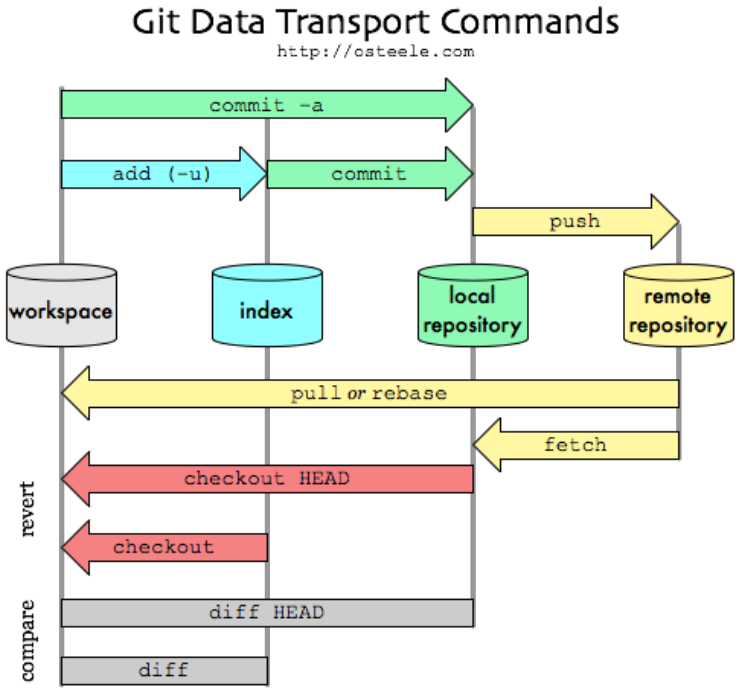


Pay attention now — here is **the main thing to remember about Git** if you want the rest of your learning process to go smoothly. **Git has three main states** that your files can reside in: **modified**, **staged**, and **committed**:

- **Modified** means that you have changed the file but have not committed it to your database yet.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed** means that the data is safely stored in your local database.

This leads us to **the three main sections of a Git project**: the working tree, the staging area, and the Git directory.

- My Git Workflow



3.2.2 Branching and Merging

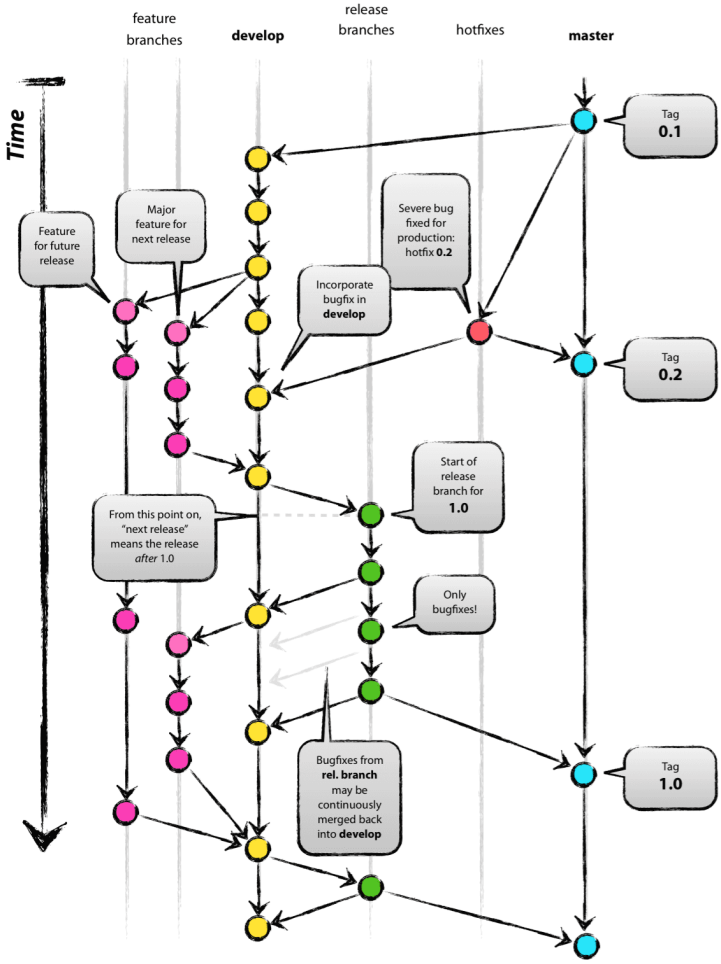
Git Branching

- Git Branching - Branches in a Nutshell

> Nearly every VCS has some form of branching support. **Branching means you diverge from the main line of development and continue to do work without messing with that main line.** In many VCS tools, this is a somewhat expensive process, often requiring you to create a new copy of your source code directory, which can take a long time for large projects.
>
> Some people refer to **Git's branching model as its "killer feature,"** and it certainly sets Git apart in the VCS community. Why is it so special? The way Git branches is incredibly lightweight, making branching operations nearly instantaneous, and switching back and forth between branches generally just as fast. Unlike many other VCSs, **Git encourages workflows that branch and merge often**, even multiple times in a day. **Understanding and mastering this feature gives you a powerful and unique tool** and can entirely change the way that you develop.

A successful Git branching model: git-flow

- A successful Git branching model (written by Vincent Driessen)



> **Note of reflection (March 5, 2020)**
>
> **This model was conceived in 2010**, now more than 10 years ago, and not very long after Git itself came into being. In those 10 years, git-flow (the branching model laid out in this article) has become hugely popular in many a software team to the point where people have started treating it

like a standard of sorts — but unfortunately also as a dogma or panacea.

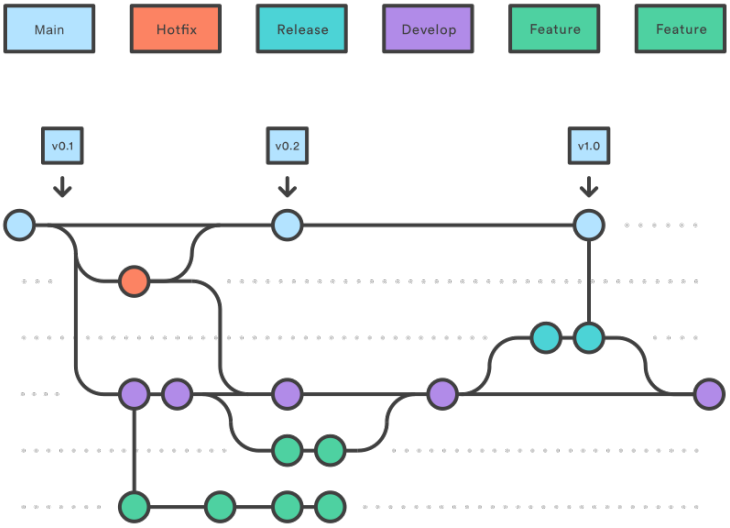
During those 10 years, Git itself has taken the world by a storm, and the most popular type of software that is being developed with Git is shifting more towards web apps — at least in my filter bubble. Web apps are typically continuously delivered, not rolled back, and you don't have to support multiple versions of the software running in the wild.

This is not the class of software that I had in mind when I wrote the blog post 10 years ago. If your team is **doing continuous delivery of software**, I would suggest to adopt **a much simpler workflow** (like GitHub flow) instead of trying to shoehorn git-flow into your team.

If, however, you are **building software that is explicitly versioned**, or if you need to **support multiple versions of your software in the wild**, then **git-flow may still be as good of a fit** to your team as it has been to people in the last 10 years. In that case, please read on.

To conclude, always remember that panaceas don't exist. Consider your own context. Don't be hating. Decide for yourself.

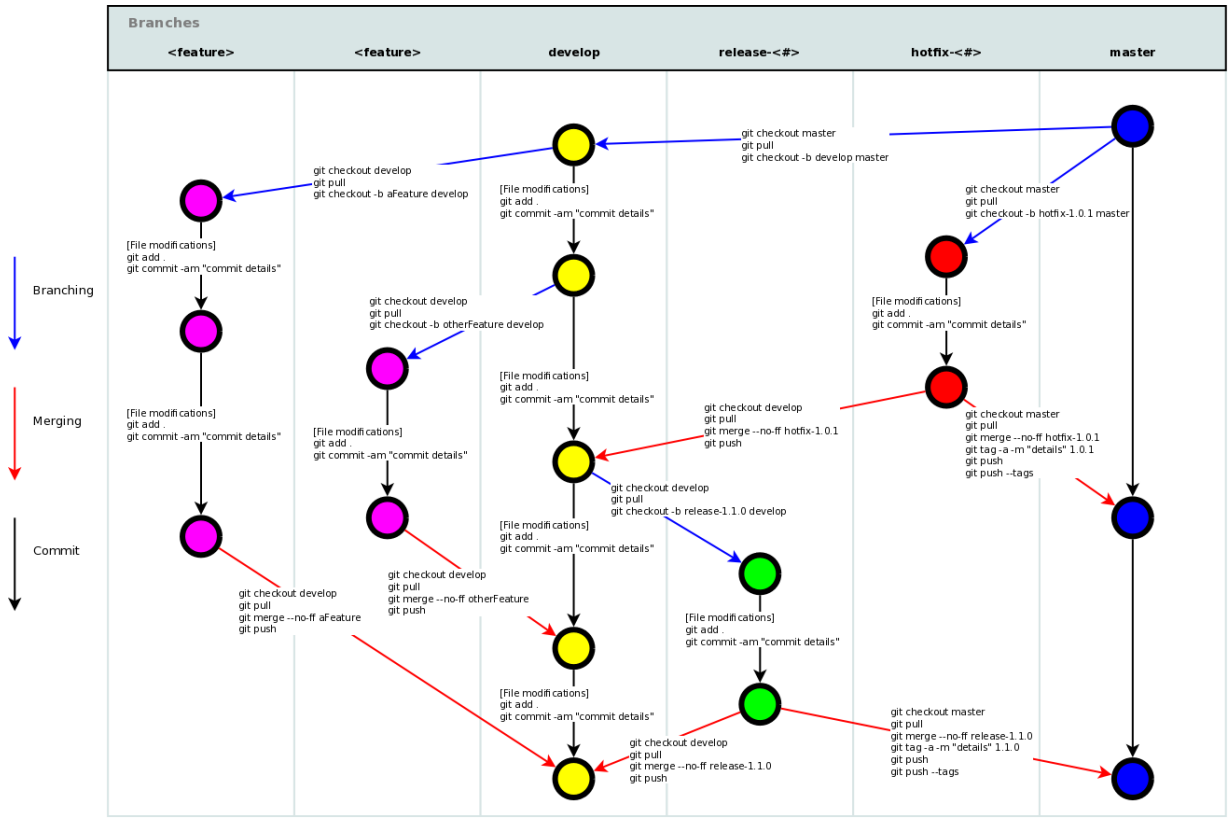
- Gitflow Workflow



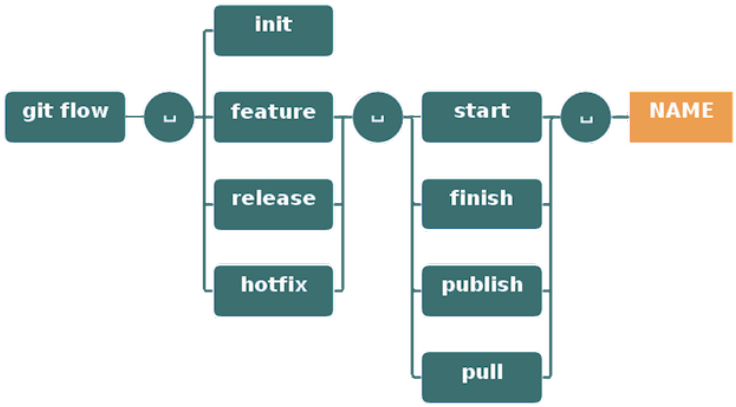
Gitflow Workflow is a Git workflow that helps with continuous software development and implementing DevOps practices. **It was first published and made popular by Vincent Driessen at nvie**. The Gitflow Workflow defines a strict branching model designed around the project release. This provides a robust framework for managing larger projects.

Gitflow is ideally suited for projects that have a scheduled release cycle and for the DevOps best practice of continuous delivery. This workflow doesn't add any new concepts or commands beyond what's required for **the Feature Branch Workflow**. Instead, it assigns very specific roles to different branches and defines how and when they should interact. In addition to **feature branches**, it uses individual branches for preparing, maintaining, and recording releases. Of course, you also get to leverage all the benefits of the Feature Branch Workflow: pull requests, isolated experiments, and more efficient collaboration.

- Git Flow Schema

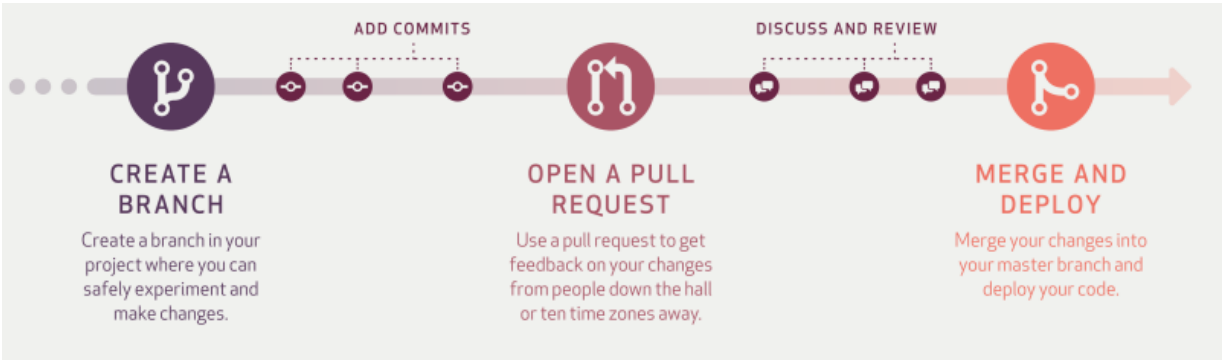


- git-flow command line tool



A much simpler workflow rather than git-flow: GitHub flow

- Understanding the GitHub flow

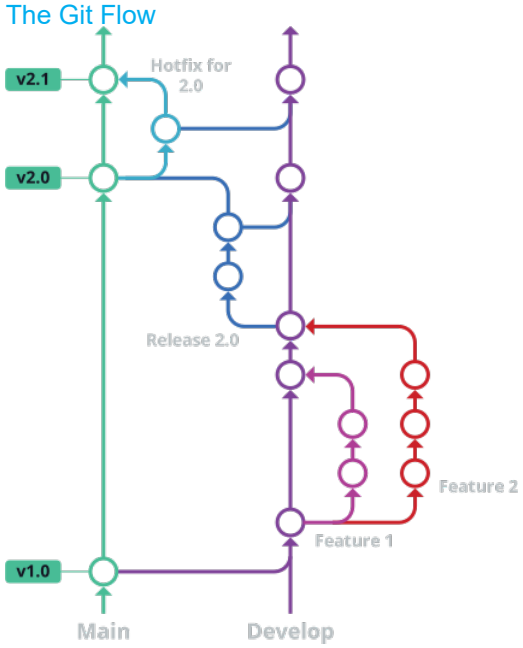


GitHub flow is **a lightweight, branch-based workflow** that supports teams and projects where **deployments are made regularly**. This guide explains how and why GitHub flow works.

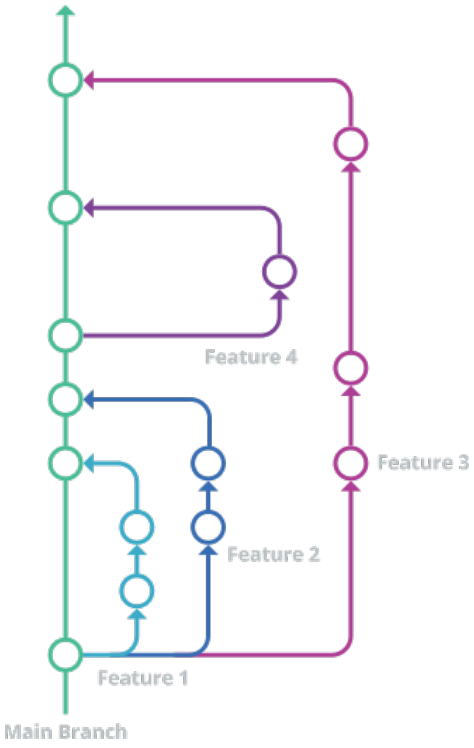
The Best Git Branch Strategy

What is the best Git branch strategy?

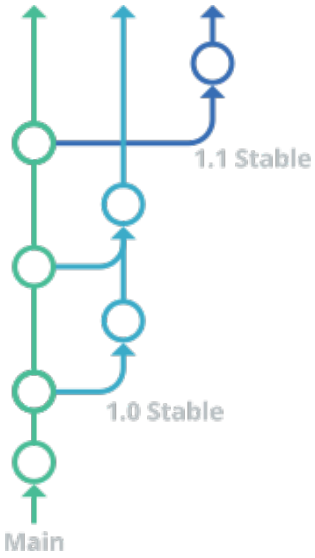
- The Git Flow



- GitHub Flow



- GitLab Flow



So, which is the best Git branching strategy?

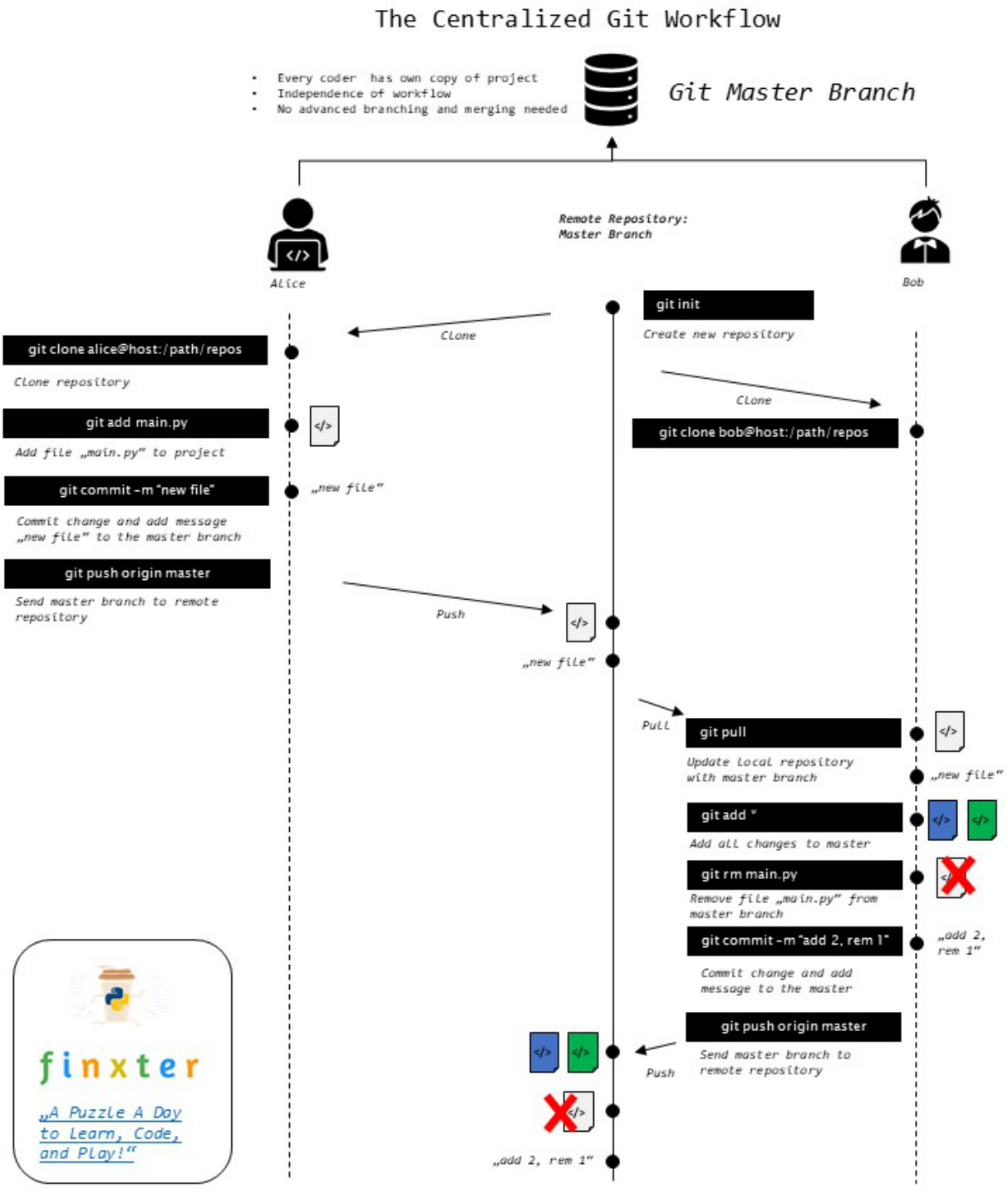
Ultimately, the answer to which Git branch strategy is **the best depends on you and your team's environment, product and your specific development needs**.

There is **not a one-size-fits-all Git branch strategy**, and regardless of which you end up selecting, it's likely you can optimize it with further modifications.

3.2.3 Simple Scenario using Git

- The Simple Git Cheat Sheet - A Helpful Illustrated Guide

The Simple Git Cheat Sheet – A Helpful Illustrated Guide



3.3 Setting up Git and Remote Repository

3.3.1 Setting up Git

Installing Git and Git Extension of VS Code on Windows

- Installing Git on Windows

3-git/setups/windows/prereqs/win-pkg-mgr.bat

```
9  @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
       InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.
       WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%
       PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

3-git/setups/windows/git/git-install.bat

```
4  choco install -y git.install
```

- Installing Git Extensions of VS Code on Windows

3-git/setups/windows/vscode/vscode-ext-git.bat

```
7  "C:\Program Files\Microsoft VS Code\bin\code" ^
8  --install-extension donjayamanne.git-extension-pack ^
9  --install-extension mhutchie.git-graph ^
10 --install-extension vector-of-bool.gitflow
```

Installing Git and Git Extension of VS Code on Ubuntu

- Installing Git on Ubuntu

3-git/setups/linux/ubuntu/git/git-install.sh

```
7  sudo apt update
8  sudo apt -y install software-properties-common
9
10 # NOTICE: select python3.8 for Ubuntu 20.04 LTS
11 sudo update-alternatives --config python3
12
13 sudo add-apt-repository -y ppa:git-core/ppa
14 sudo apt update
15 sudo apt -y install git
```

- Installing Git Extensions of VS Code on Ubuntu

3-git/setups/linux/ubuntu/vscode/vscode-ext-git.sh

```
13 if [[ -z "${SCRIPT_HOME}" ]]; then
14     SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && cd ../ && pwd -P)"
15 fi
16
17 # ============================================================================
18 # Prerequisites: code
19 # ============================================================================
20
21 . "${SCRIPT_HOME}"/cmd/cmd-exists.sh 'code'
22
23 # Git Extension Pack
24 # Popular Visual Studio Code extensions for Git
25 # https://marketplace.visualstudio.com/items?itemName=donjayamanne.git-extension-
       pack
26 #   * Git History
```

```
#     View git log, file history, compare branches or commits
#     https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory
#   * Project Manager
#     Easily switch between projects
#     https://marketplace.visualstudio.com/items?itemName=alefragnani.project-manager
#   * GitLens — Git supercharged
#     Supercharge the Git capabilities built into Visual Studio Code
#     https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens
#   * gitignore
#     Language support for .gitignore files.
#     https://marketplace.visualstudio.com/items?itemName=codezombiech.gitignore
#   * Open in GitHub, Bitbucket, Gitlab, VisualStudio.com
#     Jump to a source code line in Github, Bitbucket, Gitlab, VisualStudio.com
#     https://marketplace.visualstudio.com/items?itemName=ziyasal.vscode-open-in-github
code --install-extension donjayamanne.git-extension-pack

# Git Graph
# View a Git Graph of your repository, and perform Git actions from the graph.
# https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph
code --install-extension mhutchie.git-graph

# gitflow
# Gitflow integration and support in Visual Studio Code
# https://marketplace.visualstudio.com/items?itemName=vector-of-bool.gitflow
code --install-extension vector-of-bool.gitflow
```

Git Configuration

Getting Started - First-Time Git Setup,
Customizing Git - Git Configuration

- Configuring Git on **Windows**

`3-git/setups/windows/git/git-config.bat`

```
set USER_NAME="SIAI-Windows"
set USER_EMAIL="dev@siai.org"

:: Git User Identity
git config --global user.name "%USER_NAME%"
git config --global user.email "%USER_EMAIL%"

:: Colors in Git
git config --global color.ui "auto"

:: Formatting and Whitespace
git config --global core.autocrlf false

:: https://git-scm.com/docs/git-config#Documentation/git-config.txt-mergeff
git config --global merge.ff false

:: set main as the default branch name
:: Git 2.28.0+ for init.defaultBranch
:: https://git-scm.com/docs/git-config/2.28.0#Documentation/git-config.txt-initdefaultBranch
git config --global init.defaultBranch main

:: check your configuration settings
git config --global --list
```

- Configuring Git on **Ubuntu**

3-git/setups/linux/ubuntu/git/git-config.sh

```
16 USER_NAME_DEFAULT="SIAI-Ubuntu"
17 USER_EMAIL_DEFAULT="dev@siai.org"
18
19 USER_NAME=${1:-$USER_NAME_DEFAULT}
20 USER_EMAIL=${2:-$USER_EMAIL_DEFAULT}
21
22 # Git User Identity
23 git config --global user.name "${USER_NAME}"
24 git config --global user.email "${USER_EMAIL}"
25
26 # Colors in Git
27 git config --global color.ui "auto"
28
29 # Formatting and Whitespace
30 git config --global core.autocrlf false
31
32 # https://git-scm.com/docs/git-config#Documentation/git-config.txt-mergeff
33 git config --global merge.ff false
34
35 # set main as the default branch name
36 # Git 2.28.0+ for init.defaultBranch
37 # https://git-scm.com/docs/git-config/2.28.0#Documentation/git-config.txt-
      initdefaultBranch
38 git config --global init.defaultBranch main
39
40 # check your configuration settings
41 git config --global --list
```

3.3.2 Setting up Remote Repository

Git (Installation) Solutions and (Cloud) Services

GitHub.com, GitLab.com, BitBucket.org

- The Big Three of Git Storage: Comparing GitHub, GitLab, and Bitbucket

Features	GitHub	GitLab	Bitbucket
Free Private Repos		✔	✔
Free Public Repos	✔	✔	✔
Merge Request/Issue Templates	✔	✔	
Integrated CI	*	✔	✔
Enterprise Plans	✔	✔	✔
Integrated Project Board	✔	✔	✔
Navigation Usability	✔	✔	
Open Source		✔	
Self-hosted Option	✔	✔	✔
Discoverability	✔		
AD/LDAP	✔	✔	✔
LFS File Storage	✔	✔	✔
Integrated Time Tracking	*	✔	
Integrated Review Apps	*	✔	✔
Free, Public Static Web Pages	✔	✔	✔
Burndown Charts, Project Analytics		✔	✔
Third-party Tool Integration	✔	✔	✔

Setting up Remote Repository: GitLab.com

- Git on the Server - GitLab

> GitWeb is pretty simplistic though. If you're looking for **a modern, fully featured Git server**, there are **several open source solutions** out there that you can install instead. **As GitLab is one of the popular ones**, we'll cover installing and using it as an example. This is harder than the GitWeb option and will require more maintenance, but it is a fully featured option.

(1) **Generating and Copying your SSH Public Key**
(git-scm.com) Git on the Server - Generating Your SSH Public Key,
(gitlab.com) Generate an SSH key pair

- Creating and Copying your SSH Public Key on **Windows**

```
cd %USERPROFILE%
dir
dir .ssh :: if exists
ssh-keygen
type %USERPROFILE%\.ssh\id_rsa.pub | clip
```

- Creating and Copying your SSH Public Key on **Ubuntu**

```
cd ~/.ssh
ls
ls .ssh # if exists
ssh-keygen
cat ~/.ssh/id_rsa.pub | clip.exe # clip.exe only works on WSL
```

(2) **Signing up and in to GitLab.com**

1) Go to GitLab.com signing up page
2) Create an account using Email or Google account
3) Sign in to Gitlab.com if not signed in

(3) **Adding your SSH Public Key to Remote Repositories**
Add an SSH key to your GitLab account

```
Avatar > Preferences
> SSH Keys
> Key > {Paste your public SSH key}
> Add key
```

1) In the top right corner, select your avatar.
2) Select **Preferences**.
3) From the left sidebar, select **SSH Keys**.
4) In the **Key** box, paste the contents of your public key. If you manually copied the key, make sure you copy the entire key, which starts with `ssh-ed25519` or `ssh-rsa`, and may end with a comment.
5) In the **Title** text box, type a description, like *Work Laptop* or *Home Workstation*.
6) Select **Add key**.

(4) **Creating a blank project**
Create a project

```
+(New...) > New project/repository
> Create blank project
> Project name: Python on Linux
> Visibility Level: [X] Private
> [ ] Initialize repository with a README
> Create project
```

To create a project in GitLab:

1) In your dashboard, click the green **New project** button or use the plus icon in the navigation bar. This opens the **New project** page.
2) On the **New project** page, click **Create blank project**
3) Provide the following information:
   - The name of your project in the **Project name** field. You can't use special characters, but you can use spaces, hyphens, underscores, or even emoji. When adding the name, the **Project slug** auto populates. The slug is what the GitLab instance uses as the URL path to the project. If you want a different slug, input the project name first, then change the slug after.
   - The path to your project in the **Project slug** field. This is the URL path for your project that the GitLab instance uses. If the **Project name** is blank, it auto populates when you fill in the **Project slug**.
   - The **Project description (optional)** field enables you to enter a description for your project's dashboard, which helps others understand what your project is about. Though it's not required, it's a good idea to fill this in.
   - Changing the **Visibility Level** modifies the project's viewing and access rights for users.
   - Selecting the **Initialize repository with a README** option creates a README file so that the Git repository is initialized, has a default branch, and can be cloned.
4) Click Create project.

(5) **Adding project members to a project**

Add users to a project

```
Project Information > Members
> Project members > Invite member
> GitLab member or Email address: dev@siai.org
> Select a role: Maintainer
> Invite
```

Add users to a project so they become members and have permission to perform actions.

Prerequisite:

- You must have the **Maintainer or Owner role**.

To add a user to a project:

1. Go to your project and select **Project information > Members**.
2. On the **Invite member** tab, under **GitLab member or Email address**, type the username or email address. In GitLab 13.11 and later, you can replace this form with a modal window.

3. Select a role.
4. Optional. Choose an expiration date. On that date, the user can no longer access the project.
5. Select **Invite**.

If the user has a GitLab account, they are added to the members list. If you used an email address, the user receives an email.

If the invitation is not accepted, GitLab sends reminder emails two, five, and ten days later. Unaccepted invites are automatically deleted after 90 days.

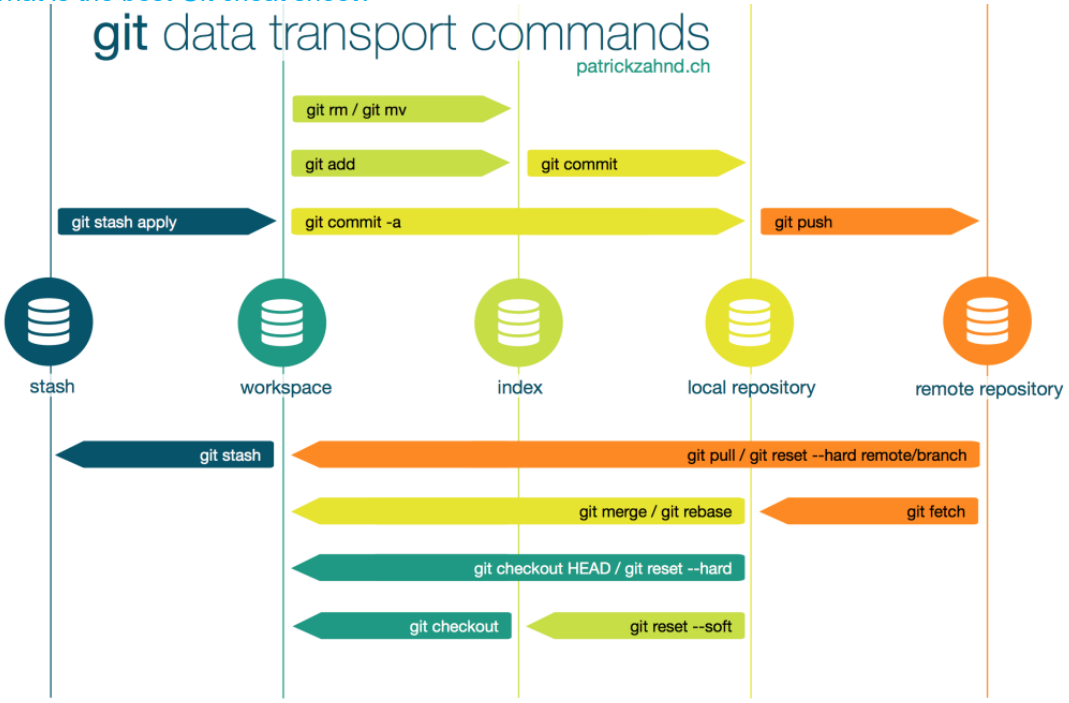
If the user does not have a GitLab account, they are prompted to create an account using the email address the invitation was sent to.

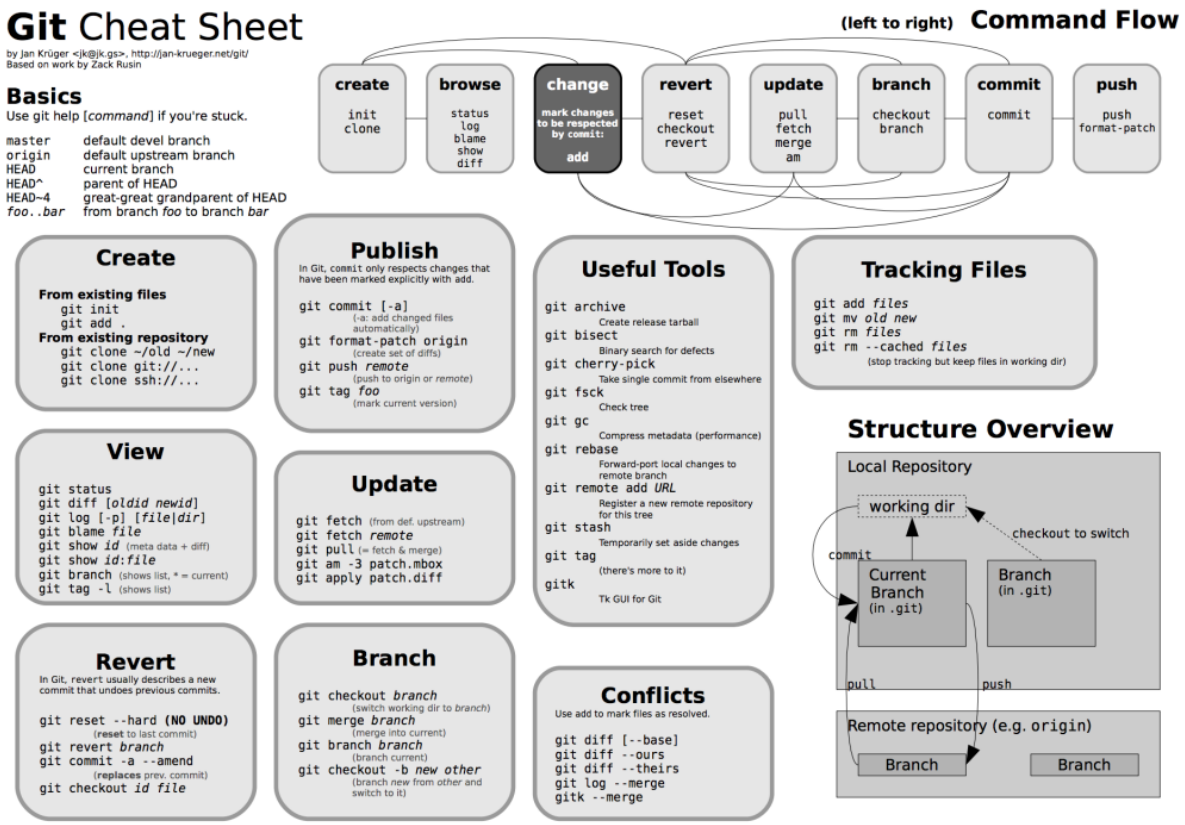
3.4 Using Git

3.4.1 Basic Git Commands

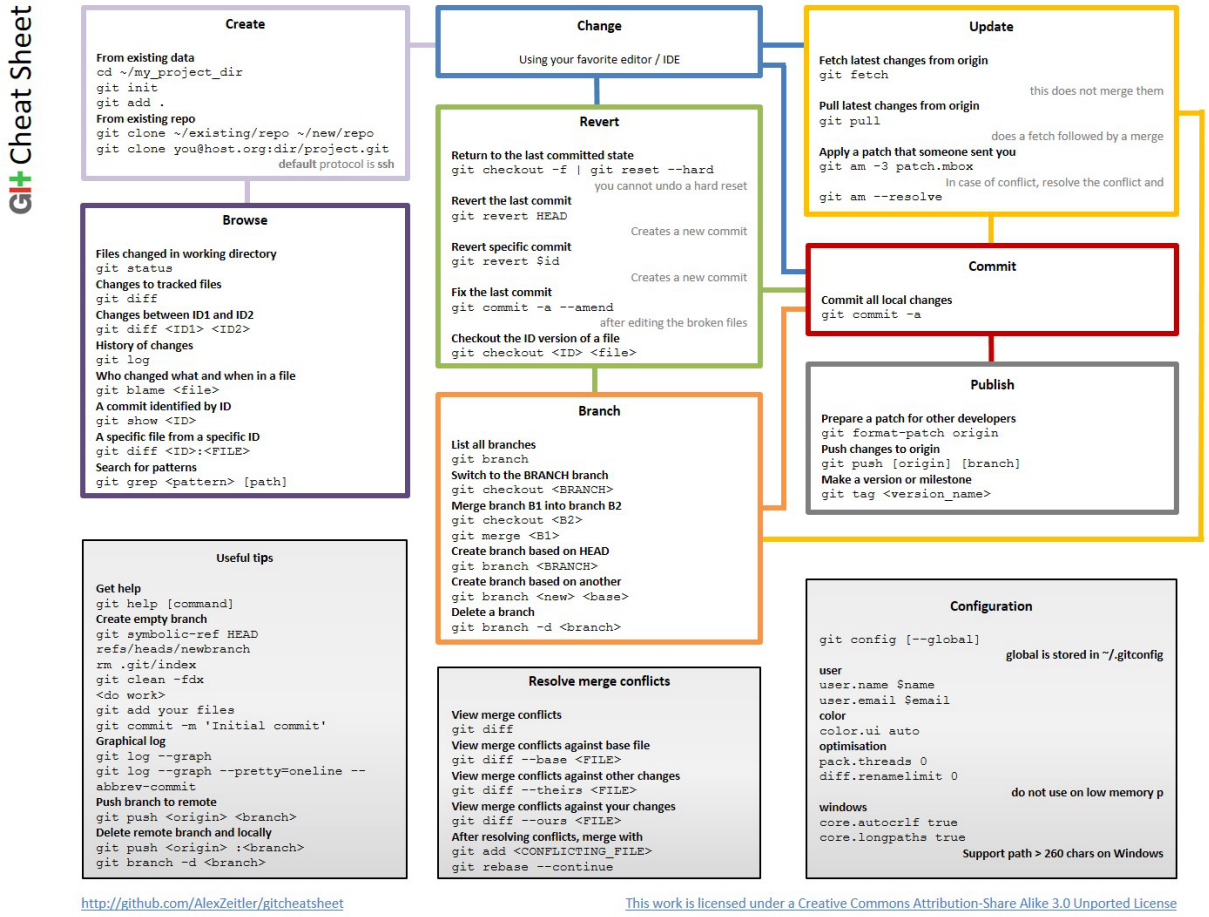
Git Cheat Sheets

- What is the best Git cheat sheet?





- AlexZeitler, gitcheatsheet



- Deleting a **Local** **Git Repository**
  on **Windows**

```
cd <repo_folder>
dir /a
rmdir /s /q .git
```

  on **Ubuntu**

```
cd <repo_folder>
ls -al
rm -rf .git
```

- Deleting a **Remote** **Git Repository**
  Delete a project on GitLab.com

> To delete a project, first navigate to the home page for that project.
>
> 1. Navigate to **Settings > General**.
> 2. Expand the **Advanced** section.
> 3. Scroll down to the **Delete project** section.
> 4. Click **Delete project**
> 5. Confirm this action by typing in the expected text.

3.4.2 Creating the `.gitignore` file for a New Git Repository

`.gitignore`

- Ignoring Files

  > Often, you'll have **a class of files that you don't want Git to automatically add or even show you as being untracked**. These are generally automatically generated files such as log files or files produced by your build system. In such cases, you can create a file listing patterns to match them named .gitignore.

- gitignore - Specifies intentionally untracked files to ignore

  > A gitignore file specifies intentionally untracked files that Git should ignore.
  > **Files already tracked by Git are not affected**.

Adding `.gitignore` in the Project Root Folder for the first time or before Uploading an Existing Folder

- gitignore extension for Visual Studio Code

```
The Command Palette (Ctrl+Shift+P or F1)
> Add gitignore
> Python

The Command Palette (Ctrl+Shift+P or F1)
> Add gitignore
> VisualStudioCode > Append

The Command Palette (Ctrl+Shift+P or F1)
> Add gitignore
> Linux > Append

The Command Palette (Ctrl+Shift+P or F1)
> Add gitignore
> Windows > Append

The Command Palette (Ctrl+Shift+P or F1)
> Add gitignore
> TeX > Append
```

`3-git/.gitignore`:

- Python

```
11 # =============================================================================
12 # Python
13 # =============================================================================
14
15 # https://github.com/github/gitignore/blob/master/Python.gitignore
16
17 # Byte-compiled / optimized / DLL files
18 __pycache__/
19 *.py[cod]
20 *$py.class
21
22 # C extensions
23 *.so
```

```
24
25 # Distribution / packaging
26 .Python
27 build/
28 develop-eggs/
29 dist/
30 downloads/
31 eggs/
32 .eggs/
33 lib/
34 lib64/
35 parts/
36 sdist/
37 var/
38 wheels/
39 share/python-wheels/
40 *.egg-info/
41 .installed.cfg
42 *.egg
43 MANIFEST
```

- Visual Studio Code

```
156 # ============================================================================
157 # Visual Studio Code
158 # ============================================================================
159
160 # https://github.com/github/gitignore/blob/master/Global/VisualStudioCode.
    gitignore
161
162 .vscode/*
163 !.vscode/settings.json
164 !.vscode/tasks.json
165 !.vscode/launch.json
166 !.vscode/extensions.json
167 *.code-workspace
```

- Linux

```
172 # ============================================================================
173 # Linux
174 # ============================================================================
175
176 # https://github.com/github/gitignore/blob/master/Global/Linux.gitignore
177
178 *~
179
180 # temporary files which can be created if a process still has a handle open of a
    deleted file
181 .fuse_hidden*
182
183 # KDE directory preferences
184 .directory
185
186 # Linux trash folder which might appear on any partition or disk
187 .Trash-*
188
189 # .nfs files are created when an open file is removed but is still being
    accessed
190 .nfs*
```

- Windows

```
# ============================================================================
# Windows
# ============================================================================

# https://github.com/github/gitignore/blob/master/Global/Windows.gitignore

# Windows thumbnail cache files
Thumbs.db
Thumbs.db:encryptable
ehthumbs.db
ehthumbs_vista.db

# Dump file
*.stackdump

# Folder config file
[Dd]esktop.ini

# Recycle Bin used on file shares
$RECYCLE.BIN/

# Windows Installer files
*.cab
*.msi
*.msix
*.msm
*.msp

# Windows shortcuts
*.lnk
```

- TeX

```
# ============================================================================
# TeX
# ============================================================================

# https://github.com/github/gitignore/blob/master/TeX.gitignore

## Core latex/pdflatex auxiliary files:
*.aux
*.lof
*.log
*.lot
*.fls
*.out
*.toc
*.fmt
*.fot
*.cb
*.cb2
.*.lb
```

3.4.3 Using Git with Commands

(Downloading once) Cloning a Remote Repository

- python-on-linux ⟵ `git@gitlab.com:siaiedu/python-on-linux.git`

```
# make workspaces folder to manage python projects
mkdir -p ~/workspaces && cd $_

```

```
git clone git@gitlab.com:siaiedu/python-on-linux.git
cd python-on-linux
# change branch
git switch -c main

# add .gitignore if not exists
touch .gitignore

touch README.md
git add README.md
git commit -m "add README"
git push -u origin main
```

(Downloading repeatedly) Pulling the Changes from the Remote Repository

- python-on-linux ⟵ `git@gitlab.com:siaiedu/python-on-linux.git`

```
git pull origin main
```

(Uploading once) Pushing a Local Repository

- `python-on-linux` ⟶ git@gitlab.com:siaiedu/python-on-linux.git

```
cd ~/workspaces/python-on-linux

# create a local git repository
# Git 2.28.0+ for --initial-branch
git init --initial-branch=main
# add a remote git repository
git remote add origin git@gitlab.com:siaiedu/python-on-linux.git

# add .gitignore if not exists
touch .gitignore

git add .
git commit -m "Initial commit"
git push -u origin main
```

(Uploading repeatedly) Pushing the Changes in the Local Repository to the Remote Repository

- `python-on-linux` ⟶ git@gitlab.com:siaiedu/python-on-linux.git

```
git push -u origin main
```

Pushing an Existing Git Repository (Changing the Remote Repository)

- python-on-linux ⟶ `git@gitlab.com:siaiedu/python-on-linux.git`

```
cd python-on-linux

git remote rename origin old-origin
git remote add origin git@gitlab.com:siaiedu/python-on-linux.git

git push -u origin --all
git push -u origin --tags
```

3.4.4 Using Git in VS Code

Official references for using Git in VS Code

- (git-scm.com) Git in Other Environments - Git in Visual Studio Code
- (code.visualstudio.com) Using Version Control in VS Code

> **Git support**
>
> **VS Code ships with a Git source control manager (SCM) extension**. Most of the source control UI and work flows are common across other SCM extensions, so reading about the general Git support in VS Code will help you understand how to use another provider.

(Uploading once) Pushing a Local Repository on Ubuntu

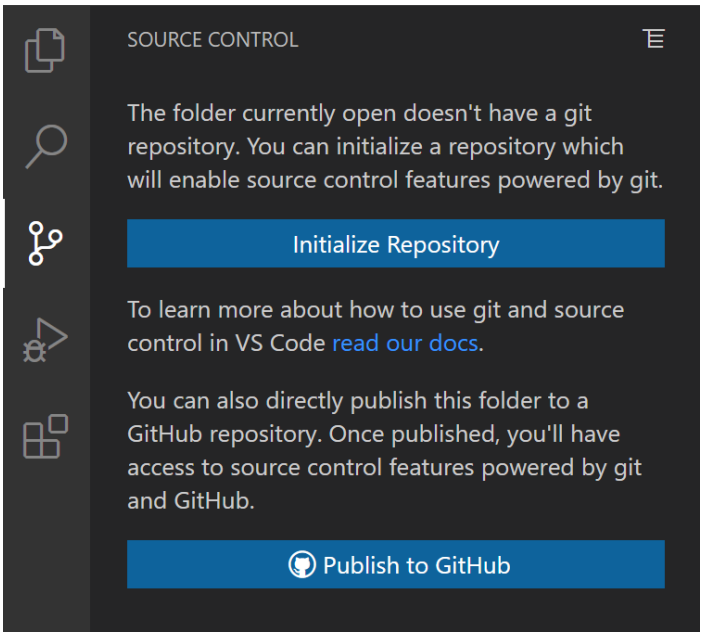
- python-on-linux ⟶ git@gitlab.com:siaiedu/python-on-linux.git

(1) `git init --initial-branch=main`

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Initialize Repository
```

Initialize a repository

> If your workspace is on your local machine, you can enable Git source control by creating a Git repository with the **Initialize Repository** command. **When VS Code doesn't detect an existing Git repository**, the Source Control view will give you the options to **Initialize Repository** or **Publish to GitHub**.



(2) `git remote add origin git@gitlab.com:siaiedu/python-on-linux.git`

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Remote > Add Remote...
> Provide repository URL or pick a repository source:
  "git@gitlab.com:siaiedu/python-on-linux.git"
> Remote name: "origin"
```

(3) `git add .`

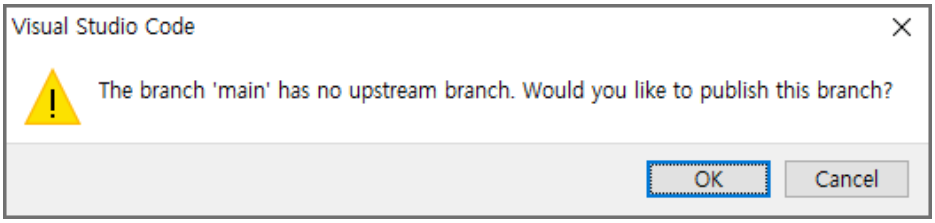
```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > Stage All Changes
```

(4) `git commit -m "Initial commit"`

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on 'main'): Initial commit
```

(5) `git push -u origin main`

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```



(6) `git log`

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

(Uploading repeatedly) Pushing the Changes in the Local Repository to the Remote Repository

- python-on-linux ⟶ git@gitlab.com:siaiedu/python-on-linux.git

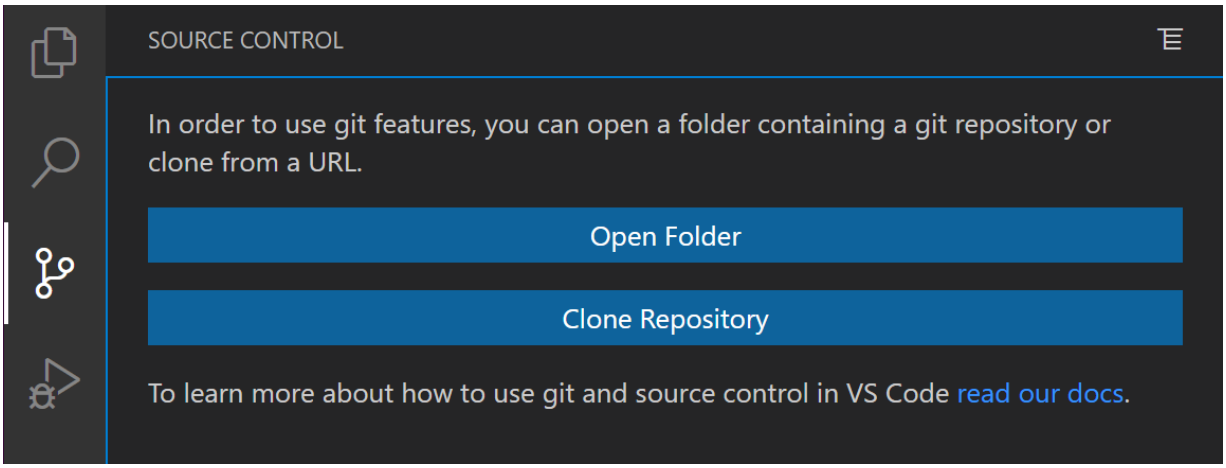
(1) `git push -u origin main`

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```

(Downloading once) Cloning a Remote Repository on Windows

- Cloning a repository

> If you **haven't opened a folder yet**, the Source Control view will give you the options to **Open Folder** from your local machine or **Clone Repository**.



- python-on-linux ⟵ git@gitlab.com:siaiedu/python-on-linux.git

(1) **`git clone git@gitlab.com:siaiedu/python-on-linux.git`**

```
Explorer (Ctrl+Shift+E)
> NO FOLDER OPENED > Clone Repository
> Provide repository URL or pick a repository source:
  "git@gitlab.com:siaiedu/python-on-linux.git"
> {Select workspace folder}
> Select Repository Location (Windows) or OK (Linux)
> Would you like to open the cloned repository? > Open
```

(2) **`git log`**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

(Downloading repeatedly) Pulling the Changes from the Remote Repository

- python-on-linux ⟵ git@gitlab.com:siaiedu/python-on-linux.git

(1) **`git pull origin main`**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Pull
```

3.4.5 Resolving a Merge Conflict in VS Code

Understanding Why a Merge Conflict Occurs

- Basic Merge Conflicts

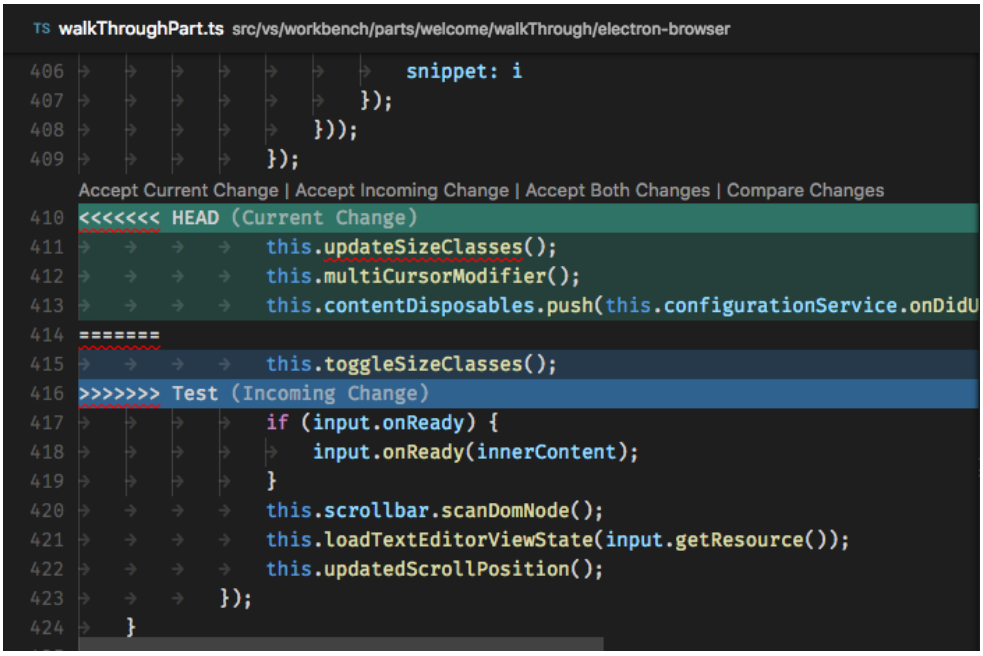
> If you **changed the same part of the same file differently** in the two branches you're merging, Git won't be able to merge them cleanly.

- Git Tools - Advanced Merging

> Merging in Git is typically fairly easy. Since Git makes it easy to merge another branch multiple times, it means that you can have a very long lived branch but you can keep it up to date as you go, **solving small conflicts often**, rather than be surprised by one enormous conflict at the end of the series.
>
> However, sometimes tricky conflicts do occur. Unlike some other version control systems, Git does not try to be overly clever about merge conflict resolution. **Git's philosophy is to be smart about determining when a merge resolution is unambiguous, but if there is a conflict, it does not try to be clever about automatically resolving it**. Therefore, if you wait too long to merge two branches that diverge quickly, you can run into some issues.

```
def hello:
<<<<<<< HEAD
    print('hola world')
=======
    print('hello mundo')
>>>>>>> mundo

hello()
```

- Merge conflicts



Changing the same part of the same file differently (a merge conflict occurs)

(1) Changing the Hello message in `2-python-on-linux/src/hello.py` on **Windows**

```
msg = "hello mundo"
print(msg)
```

(2) Adding, committing, and pushing the source code on **Windows**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > hello.py > Stage Changes

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on 'main'): update hello message
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

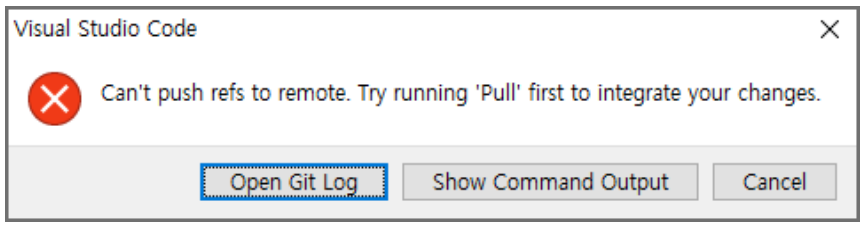
(3) Changing the Hello message in `2-python-on-linux/src/hello.py` on **Ubuntu**

```
msg = "hola world"
print(msg)
```

(4) Adding, committing, and pushing the source code on **Ubuntu**

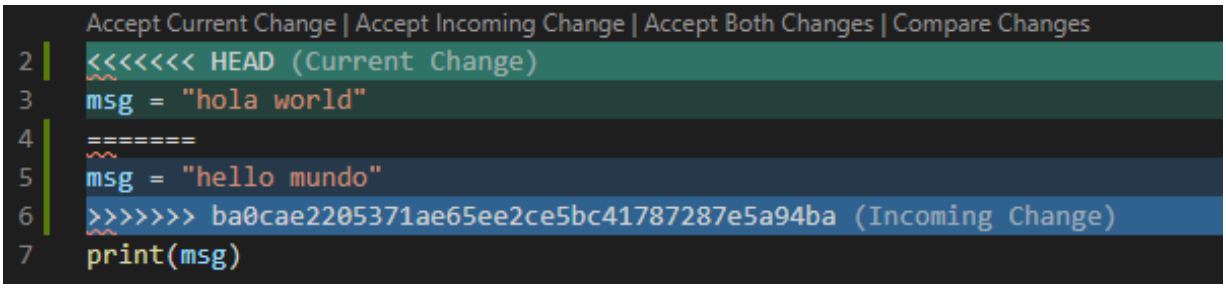
```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > hello.py > Stage Changes

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on 'main'): update hello message
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```



(5) Pulling the repository on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Pull
```



(6) Resolving a Merge Conflict on **Ubuntu**:
accepting change(s), committing, and pushing the source code

1) **Accept Current Change** | **Accept Incoming Change** | **Accept Both Changes**

```
<<<<<<< HEAD (Current Change)
msg = "hola world"
=======
msg = "hello mundo"
>>>>>>> ba0cae2205371ae65ee2ce5bc41787287e5a94ba (Incoming Change)
print(msg)
```

- **Accept Current Change**

```
msg = "hola world"
print(msg)
```

- **Accept Incoming Change**

```
msg = "hello mundo"
print(msg)
```

- **Accept Both Changes**

```
msg = "hola world"
msg = "hello mundo"
print(msg)
```

2) Staging the conflicting file, committing, and pushing those changes

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Merge Changes > hello.py > Stage Changes

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
(> "There are no changes to commit" > "Create Empty Commit")

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions... > Push
```

NOTICE: If you chose the "**Accept Current Change**", you should **Create Empty Commit**.
Refer to Visual Studio Code doesn't stage Git merge changes for "**Create Empty Commit**"

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

(7) Pulling the repository on **Windows**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Pull
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

Changing the different part of the same file respectively

(1) Adding print function in `2-python-on-linux/src/hello.py` on **Windows**

```python
print("Hello World")  # windows
msg = "hola world"
print(msg)
```

(2) Adding, committing, and pushing the source code on **Windows**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > Stage All Changes
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on '{branch name}'): {commit message}
```

- branch name: main
- commit message: **add print from windows**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

(3) Adding print function in `2-python-on-linux/src/hello.py` on **Ubuntu**

```python
msg = "hola world"
print(msg)  # ubuntu
print(msg)
```

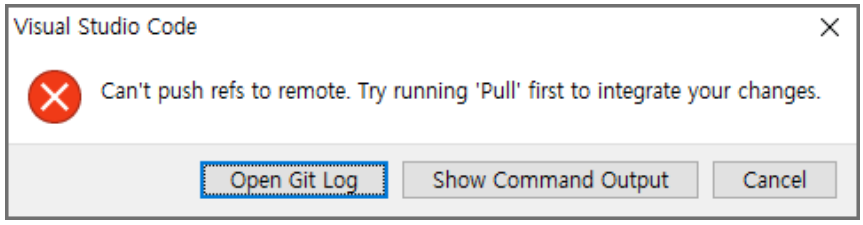
(4) Adding, committing, and pushing the source code on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > Stage All Changes
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on '{branch name}'): {commit message}
```

- branch name: main
- commit message: **add msg from ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```




```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

(5) Pulling and pushing commits from and to the remote repository on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Pull
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

Changing the different files respectively

(1) Changing code in `2-python-on-linux/src/iris_data_1.py` on **Windows** as following:

```
# X = iris.data
# y = iris.target
X, y = iris.data, iris.target
```

(2) Adding, committing, and pushing the source code on **Windows**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > Stage All Changes
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on '{branch name}'): {commit message}
```

- branch name: main
- commit message: **change code on windows**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

(3) Changing code in `2-python-on-linux/src/iris_data_7.py` on **Ubuntu** as following:

```
IRIS_DATA_URL = (
    "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
)

df = pd.read_csv(IRIS_DATA_URL)
```

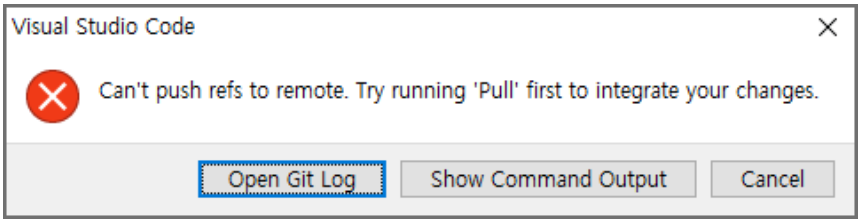
(4) Adding, committing, and pushing the source code on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > Stage All Changes
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on '{branch name}'): {commit message}
```

- branch name: main
- commit message: **change code on ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```



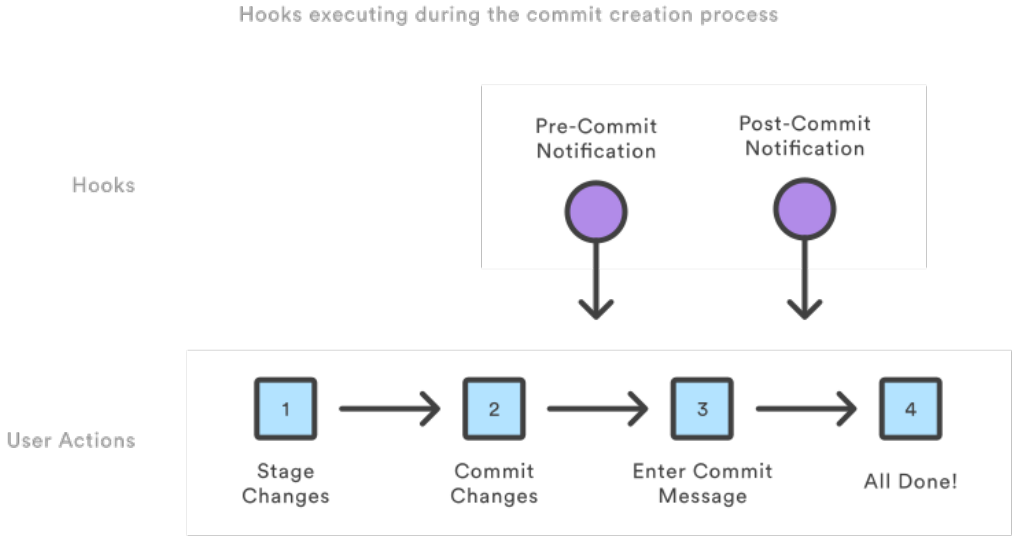
(5) Pulling and pushing commits to and from the remote repository on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Pull, Push > Sync
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Git: View History (git log)
> Refresh

Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > View Git Graph (git log)
```

Considering Git (Client-Side) Hooks for Commit Policies

- (Bitbucket) Git Hooks



- Customizing Git - Git Hooks

> **Git Hooks**
>
> Like many other Version Control Systems, Git has a way to **fire off custom scripts when certain important actions occur**. There are **two groups of these hooks: client-side and server-side**. **Client-side hooks** are **triggered by operations such as committing and merging**, while

> **server-side hooks** run on **network operations such as receiving pushed commits**. You can use these hooks for all sorts of reasons.

- Working with Git commit policies
- Working with GitLab.com commit policies
- Working with GitHub commit policies
- Working with Bitbucket Server commit policies

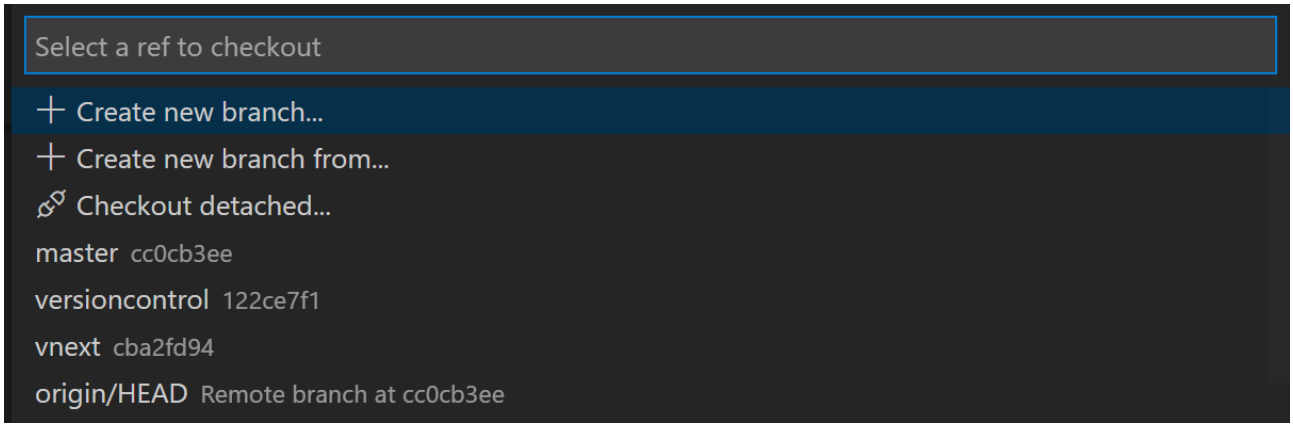
3.4.6 Using Git Branches and Gitflow in VS Code

Git Branches in VS Code

Branches and Tags

You can create and checkout branches directly within VS code through the **Git: Create Branch** and **Git: Checkout to** commands in the **Command Palette** (`Ctrl+Shift+P`).

If you run **Git: Checkout to**, you will see a dropdown list containing all of the branches or tags in the current repository. It will also give you the option to create a new branch if you decide that's a better option, or checkout a branch in detached mode.



Using Git Branches in VS Code on Ubuntu

(1) Creating and switching to `develop` branch

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Branch > Create Branch...
> Branch name > develop
```

(2) Merging `main` branch into `develop` branch

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Branch > Merge Branch...
> Select a branch to merge from > {branch name}
```

- branch name: **main**

(3) Changing code in `2-python-on-linux/src/iris_data_7.py` on **Ubuntu** as following:

```
df = pd.read_csv(IRIS_DATA_URL)

print(df)  # develop branch on ubuntu
print(df.shape)
```

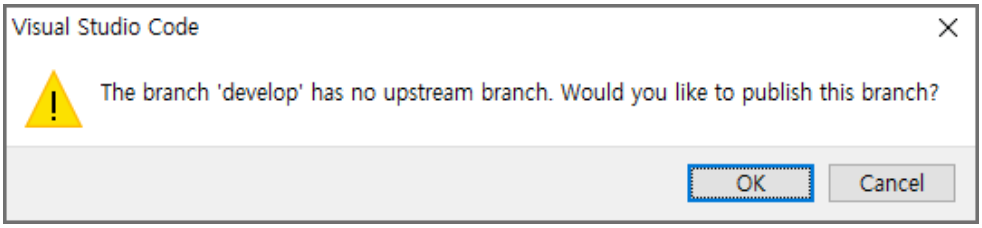
(4) Adding, committing, and pushing the source code on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Changes > Stage All Changes
```

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > Commit
> Message (commit on '{branch name}'): {commit message}
```

- branch name: develop
- commit message: **change code on develop branch on ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```



(5) Switching to the `main` branch

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Checkout to...
> Select a ref to checkout > {branch name}
```

- branch name: **main**

(6) Merging `develop` branch into `main` branch

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Branch > Merge Branch...
> Select a branch to merge from > {branch name}
```

- branch name: **develop**

(7) Pushing the commit to the remote repository on **Ubuntu**

```
Source Control (Ctrl+Shift+G)
> SOURCE CONTROL > More Actions...
> Push
```

3.4.7 Git Commit Policies

Recommended Git Commit Policies

- commit a single file if possible
- commit early, push often
- pull before push
- pull before commit

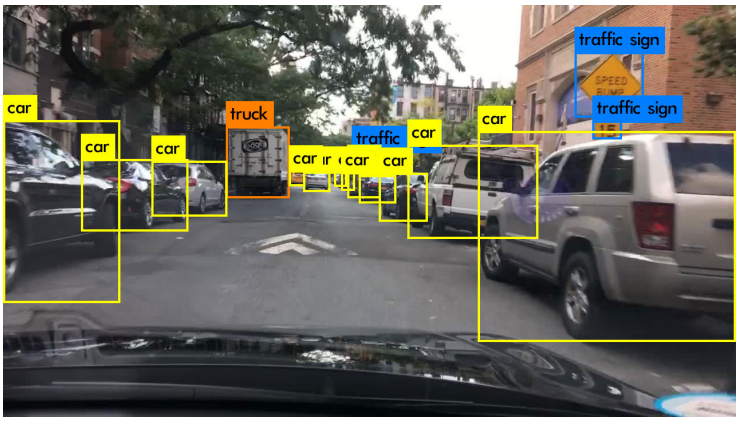
References for Git Commit Policies

- 5 Git Best Practices For Git Commit
  - Branch Frequently, Commit Often
  - Make Small, Single-Purpose Commits
  - Write Short, Detailed Commit Messages
  - Test Code and Require Reviews
  - Preserve History and Traceability

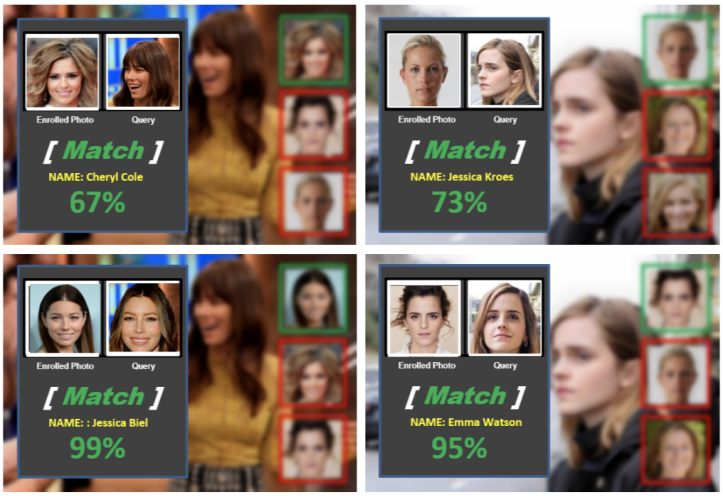
3.4.8 GitHub.com Open Projects for Data Science

Computer Vision (CV)

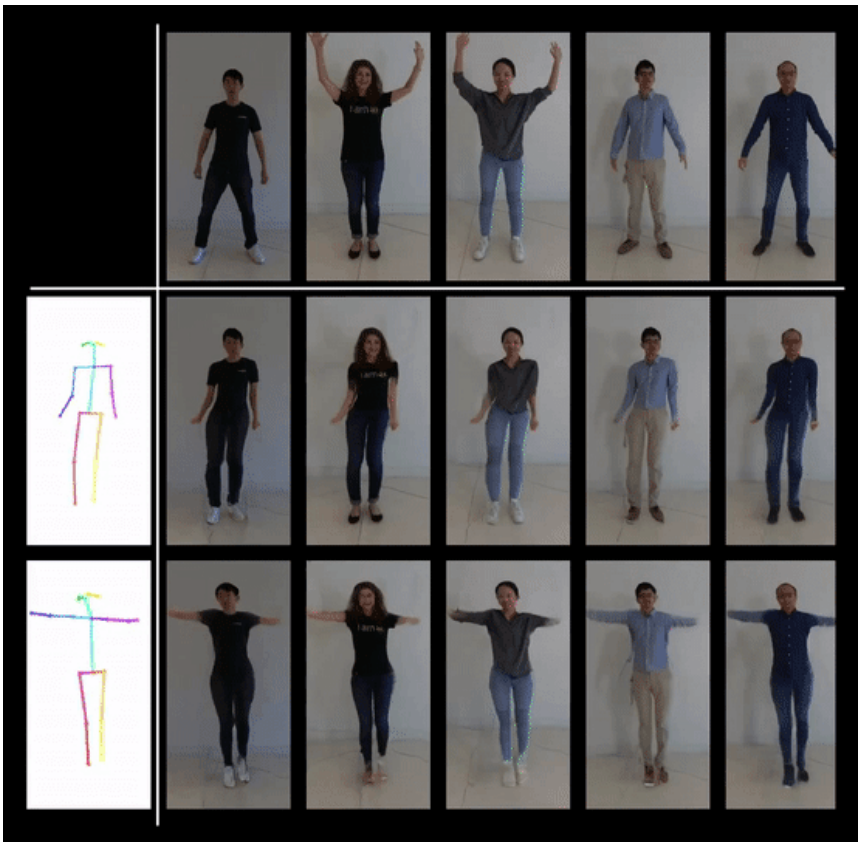
- Gaussian YOLOv3: An Accurate and Fast Object Detector Using Localization Uncertainty for Autonomous Driving



- face.evoLVe: High-Performance Face Recognition Library based on PyTorch & PaddlePaddle

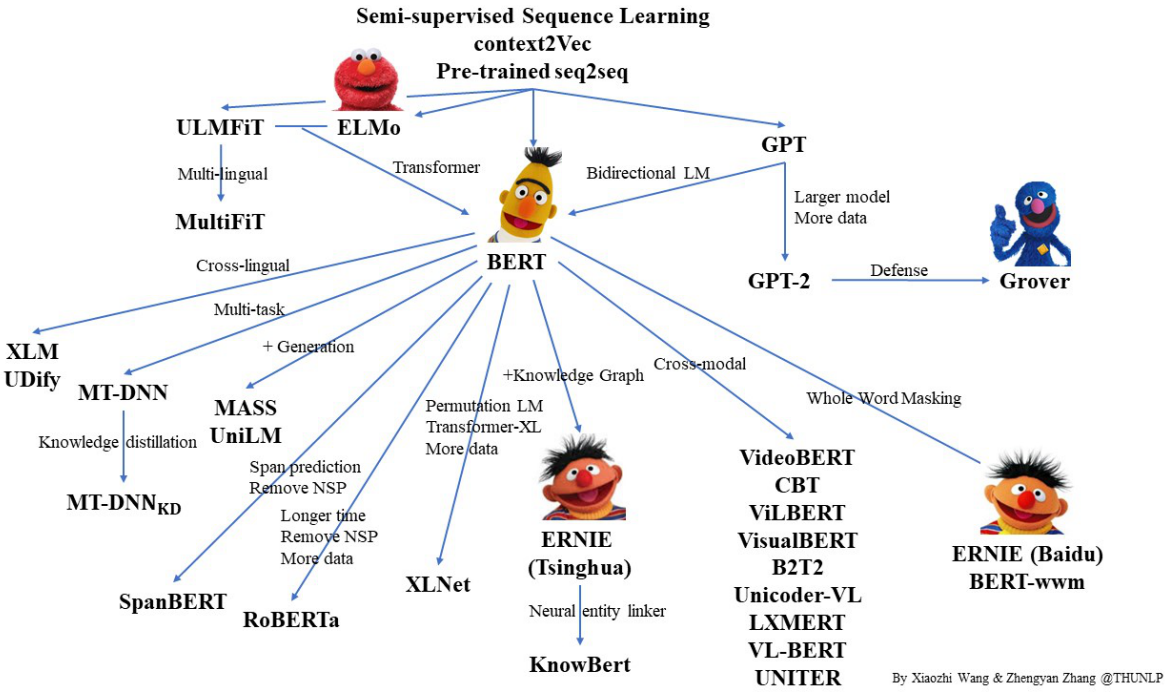


- Few-shot vid2vid (video-to-video translation)



Natural Language Processing (NLP)

- Must-Read Papers on Pre-trained Language Models (PLMs)



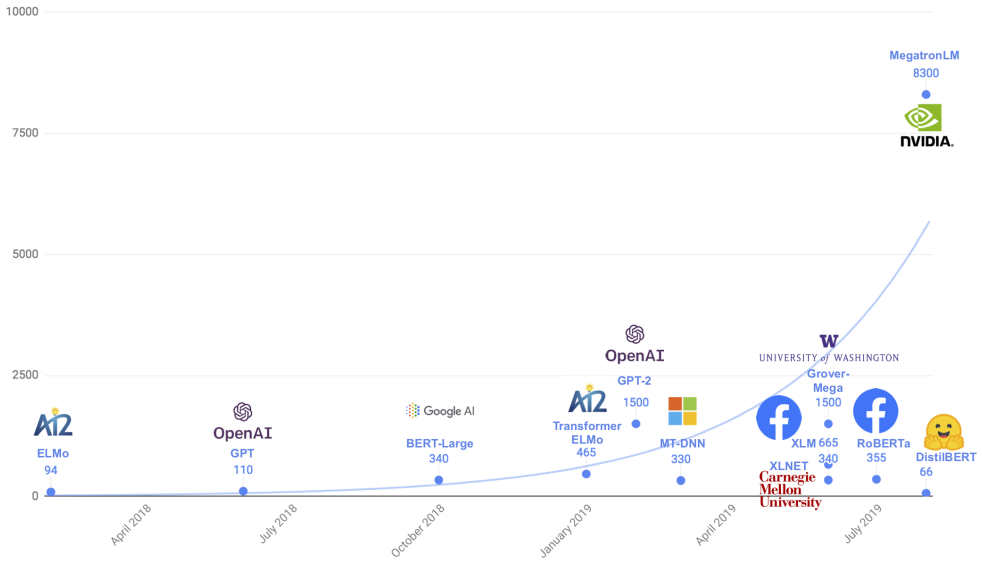
> Pre-trained Languge Model (PLM) has achieved great success in NLP since 2018. In this repo, we list some representative work on PLMs and show their relationship with a diagram.

- DistilBERT – A Lighter and Cheaper Version of Google's BERT

- (GitHub) huggingface/transformers

  This folder contains the original code used to train Distil* as well as examples showcasing how to use DistilBERT, DistilRoBERTa and DistilGPT2.

Parameter counts of several recently released pretrained language models



- **gpt-2-simple**

  A simple Python package that wraps existing model fine-tuning and generation scripts for OpenAI's GPT-2 text generation model (specifically the "small" 124M and "medium" 355M hyperparameter versions).

TODO: Clone the repository and view the Git Graph

3.5 Automation for using Git

3.5.1 Setting up Git

Setting up Git on Windows

(1) **Installing Git:**

`3-git/setups/windows/git/git-install.bat`

```
@echo off

:: https://chocolatey.org/packages/git.install
choco install -y git.install
```

(2) **Configuring Git:**

`3-git/setups/windows/git/git-config.bat`

```
@echo off

:: ########################################################################################
:: Git configurations
:: ########################################################################################

:: 1.6 Getting Started - First-Time Git Setup
:: https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

:: 8.1 Customizing Git - Git Configuration
:: https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration

:: git-config
:: https://git-scm.com/docs/git-config

set USER_NAME="SIAI-Windows"
set USER_EMAIL="dev@siai.org"

:: Git User Identity
git config --global user.name "%USER_NAME%"
git config --global user.email "%USER_EMAIL%"

:: Colors in Git
git config --global color.ui "auto"

:: Formatting and Whitespace
git config --global core.autocrlf false

:: https://git-scm.com/docs/git-config#Documentation/git-config.txt-mergeff
git config --global merge.ff false

:: set main as the default branch name
:: Git 2.28.0+ for init.defaultBranch
:: https://git-scm.com/docs/git-config/2.28.0#Documentation/git-config.txt-
    initdefaultBranch
git config --global init.defaultBranch main

:: check your configuration settings
git config --global --list
```

(3) **Installing VS Code Extensions for Git:**

`3-git/setups/windows/vscode/vscode-ext-git.bat`

```
@echo off

```

```
:: ###########################################################################
:: Installing Extensions of Visual Studio Code (VS Code)
:: ###########################################################################

"C:\Program Files\Microsoft VS Code\bin\code" ^
--install-extension donjayamanne.git-extension-pack ^
--install-extension mhutchie.git-graph ^
--install-extension vector-of-bool.gitflow
```

(4) **Setting up Git on Windows:**

`3-git/setups/windows/git-setup.bat`

```
@echo off

pushd "%~dp0"

set SCRIPT_HOME=%cd%

call "%SCRIPT_HOME%\git\git-install.bat"
call "%SCRIPT_HOME%\git\git-config.bat"
call "%SCRIPT_HOME%\vscode\vscode-ext-git.bat"
```

Setting up Git on Ubuntu

(1) **Installing Git:**

`3-git/setups/linux/ubuntu/git/git-install.sh`

```
#!/usr/bin/env bash

# ###########################################################################
# Installing Git on Ubuntu
# ###########################################################################

sudo apt update
sudo apt -y install software-properties-common

# NOTICE: select python3.8 for Ubuntu 20.04 LTS
sudo update-alternatives --config python3

sudo add-apt-repository -y ppa:git-core/ppa
sudo apt update
sudo apt -y install git
```

(2) **Configuring Git:**

`3-git/setups/linux/ubuntu/git/git-config.sh`

```
#!/usr/bin/env bash

# ###########################################################################
# Git configurations
# ###########################################################################

# 1.6 Getting Started - First-Time Git Setup
# https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

# 8.1 Customizing Git - Git Configuration
# https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration

# git-config
# https://git-scm.com/docs/git-config

```

```
16 USER_NAME_DEFAULT="SIAI-Ubuntu"
17 USER_EMAIL_DEFAULT="dev@siai.org"
18
19 USER_NAME=${1:-$USER_NAME_DEFAULT}
20 USER_EMAIL=${2:-$USER_EMAIL_DEFAULT}
21
22 # Git User Identity
23 git config --global user.name "${USER_NAME}"
24 git config --global user.email "${USER_EMAIL}"
25
26 # Colors in Git
27 git config --global color.ui "auto"
28
29 # Formatting and Whitespace
30 git config --global core.autocrlf false
31
32 # https://git-scm.com/docs/git-config#Documentation/git-config.txt-mergeff
33 git config --global merge.ff false
34
35 # set main as the default branch name
36 # Git 2.28.0+ for init.defaultBranch
37 # https://git-scm.com/docs/git-config/2.28.0#Documentation/git-config.txt-
      initdefaultBranch
38 git config --global init.defaultBranch main
39
40 # check your configuration settings
41 git config --global --list
```

(3) **Installing VS Code Extensions for Git:**

`3-git/setups/linux/ubuntu/vscode/vscode-ext-git.sh`

```
1  #!/usr/bin/env bash
2
3  # ##############################################################################
4  # Installing Extensions of Visual Studio Code (VS Code) for Git
5  # ##############################################################################
6
7  # Using Version Control in VS Code: Git support
8  # https://code.visualstudio.com/docs/editor/versioncontrol#_git-support
9
10 # Git version control in VS Code
11 # https://code.visualstudio.com/docs/introvideos/versioncontrol
12
13 if [[ -z "${SCRIPT_HOME}" ]]; then
14     SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && cd ../ && pwd -P)"
15 fi
16
17 # ==============================================================================
18 # Prerequisites: code
19 # ==============================================================================
20
21 . "${SCRIPT_HOME}"/cmd/cmd-exists.sh 'code'
22
23 # Git Extension Pack
24 # Popular Visual Studio Code extensions for Git
25 # https://marketplace.visualstudio.com/items?itemName=donjayamanne.git-extension-
      pack
26 #    * Git History
27 #      View git log, file history, compare branches or commits
28 #      https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory
29 #    * Project Manager
30 #      Easily switch between projects
```

```
#     https://marketplace.visualstudio.com/items?itemName=alefragnani.project-manager
#   * GitLens — Git supercharged
#     Supercharge the Git capabilities built into Visual Studio Code
#     https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens
#   * gitignore
#     Language support for .gitignore files.
#     https://marketplace.visualstudio.com/items?itemName=codezombiech.gitignore
#   * Open in GitHub, Bitbucket, Gitlab, VisualStudio.com
#     Jump to a source code line in Github, Bitbucket, Gitlab, VisualStudio.com
#     https://marketplace.visualstudio.com/items?itemName=ziyasal.vscode-open-in-github
code --install-extension donjayamanne.git-extension-pack

# Git Graph
# View a Git Graph of your repository, and perform Git actions from the graph.
# https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph
code --install-extension mhutchie.git-graph

# gitflow
# Gitflow integration and support in Visual Studio Code
# https://marketplace.visualstudio.com/items?itemName=vector-of-bool.gitflow
code --install-extension vector-of-bool.gitflow
```

(4) **Setting up Git on Ubuntu:**

`3-git/setups/linux/ubuntu/git-setup.sh`

```
#!/usr/bin/env bash

SCRIPT_HOME="$(cd "$(dirname "${BASH_SOURCE}")" && pwd -P)"

. "${SCRIPT_HOME}/git/git-install.sh"
. "${SCRIPT_HOME}/git/git-config.sh"
. "${SCRIPT_HOME}/vscode/vscode-ext-git.sh"
```

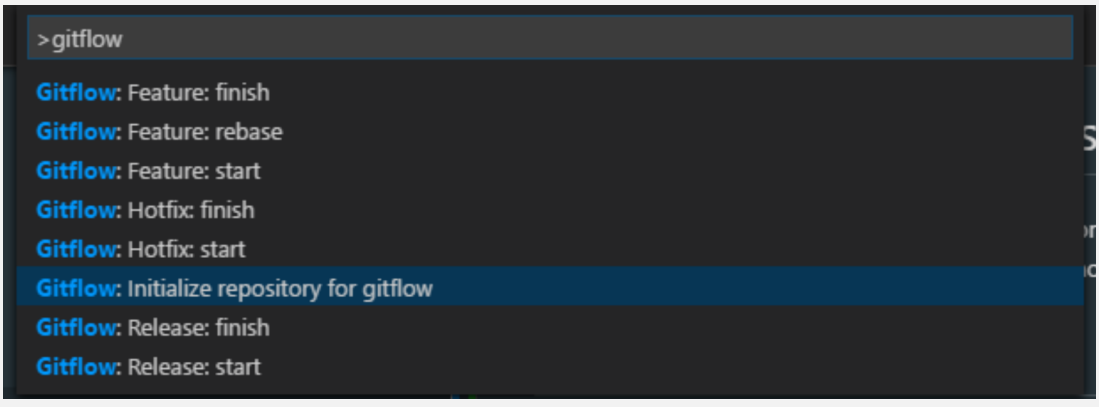
3.5.2 Using Git

Using Gitflow in VS Code

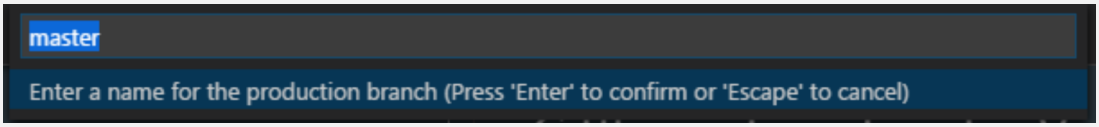
- Gitflow integration for Visual Studio Code

Starting from Scratch

1. First, initialize git: `git init`
2. Open the VS Code Command Palette and type 'gitflow'
3. Select **'Initialize repository for gitflow'**



4. Follow the command prompts and accept the defaults...



5. Setup complete!

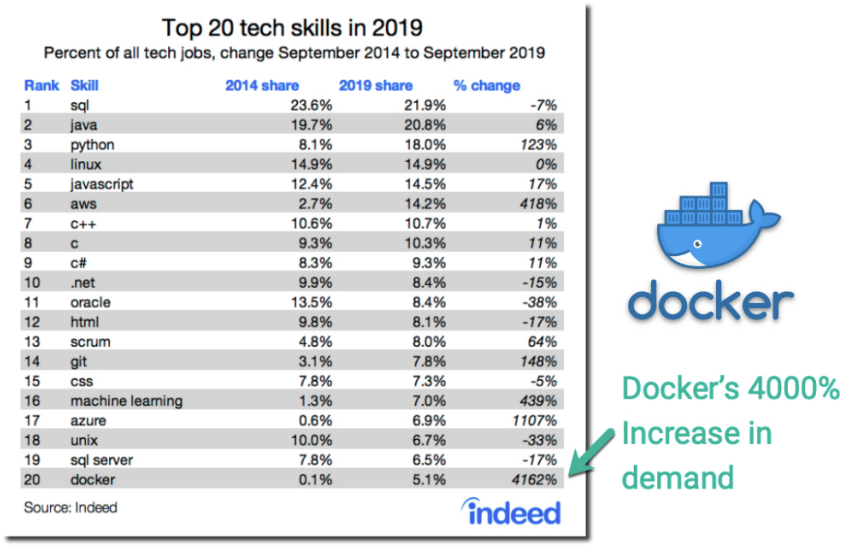
Chapter 4

Sharing Your Python Development Environment and Runtime with a Docker Container

4.1 Why Docker for Data Science?

4.1.1 A Top Skill for 2020

- Part 3 - Docker for Data Scientists (A Top Skill for 2020)



Top 20 tech skills in 2019

Percent of all tech jobs, change September 2014 to September 2019

Rank	Skill	2014 share	2019 share	% change
1	sql	23.6%	21.9%	-7%
2	java	19.7%	20.8%	6%
3	python	8.1%	18.0%	123%
4	linux	14.9%	14.9%	0%
5	javascript	12.4%	14.5%	17%
6	aws	2.7%	14.2%	418%
7	c++	10.6%	10.7%	1%
8	c	9.3%	10.3%	11%
9	c#	8.3%	9.3%	11%
10	.net	9.9%	8.4%	-15%
11	oracle	13.5%	8.4%	-38%
12	html	9.8%	8.1%	-17%
13	scrum	4.8%	8.0%	64%
14	git	3.1%	7.8%	148%
15	css	7.8%	7.3%	-5%
16	machine learning	1.3%	7.0%	439%
17	azure	0.6%	6.9%	1107%
18	unix	10.0%	6.7%	-33%
19	sql server	7.8%	6.5%	-17%
20	docker	0.1%	5.1%	4162%

Source: Indeed

indeed

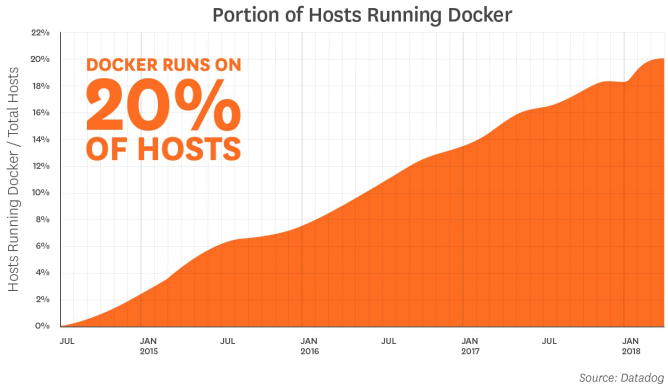
> Indeed, the popular employment-related search engine, released an article this past Tuesday showing changing trends from 2015 to 2019 in "Technology-Related Job Postings". We can see a number of changes in key technologies - One that we are particularly interested in is the 4000% increase in Docker.

4.1.2 Why Linux for Data Science ⟶ Why Docker for Data Science

- (Production Environments) Mostly Linux Servers ⟶ the Share of Hosts running Docker has increased
- (Development Environments) Linux only Data Science Software ⟶ Dockerized Data Science Software

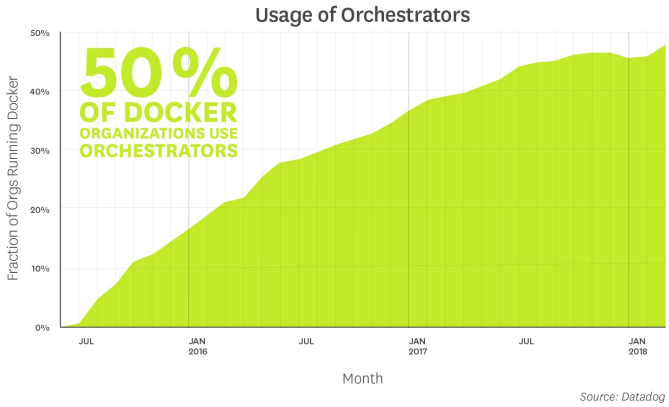
(Production Environments) Mostly Linux Servers $\longrightarrow$ the Share of Hosts running Docker has increased

- 8 surprising facts about real Docker adoption - 2018
  - **Docker Now Runs on More Than 20% of Hosts**



> Across all the environments monitored by Datadog, **the share of hosts running Docker also continues to climb**. Approximately 21 percent of all hosts now run Docker, as of April 2018. **That share has increased** at a steady clip of about 5 points per year.

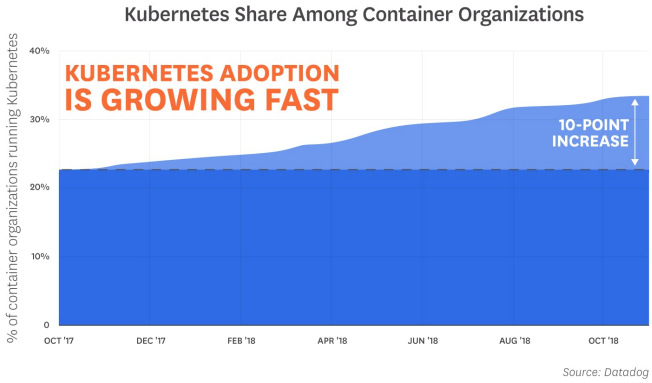
  - **Half of Docker Environments Are Orchestrated** [1]



> Increasingly, Docker is not being run as a standalone technology but as part of a larger containerization strategy, which includes automated orchestration of workloads. Roughly half of the companies that monitor Docker with Datadog now also **monitor an orchestrator such as Kubernetes or Mesos**, or a hosted orchestration platform from AWS, Azure, or Google Cloud Platform.

- 8 emerging trends in container orchestration - 2018
  - **Kubernetes continues its steady rise in container environments globally**

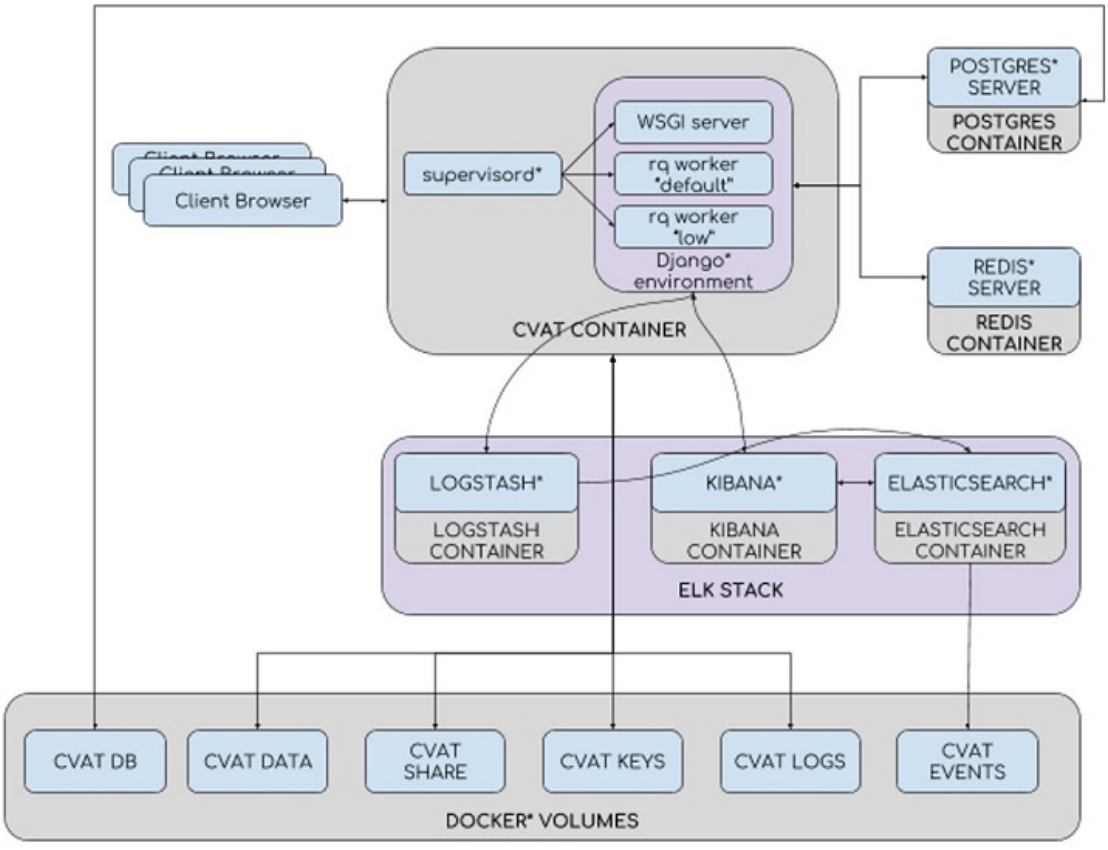
[1] Pets (management) vs. Cattle (orchestration)



Kubernetes is everywhere. Not only do our customers run their own Kubernetes clusters in diverse infrastructure environments, but all three major cloud providers now offer a managed Kubernetes service as well. One third of our customers using containers now use Kubernetes, whether in self-managed clusters, or through a cloud service like Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), or the new Amazon Elastic Container Service for Kubernetes (EKS).

(Development Environments) Linux only Data Science Software ⟶ Dockerized Data Science Software

- **Computer Vision (CV)**
  - Computer Vision Annotation Tool: A Universal Approach to Data Annotation
    The Internal Structure

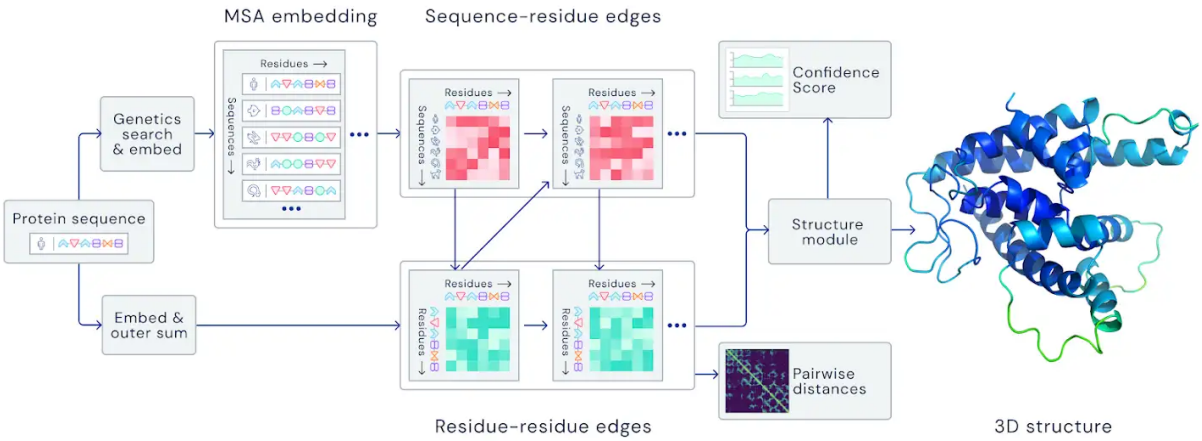


- **Natural Language Processing (NLP)**
  - Kakao Hangul Analyzer III (khaiii)

khaiii/docker/Dockerfile

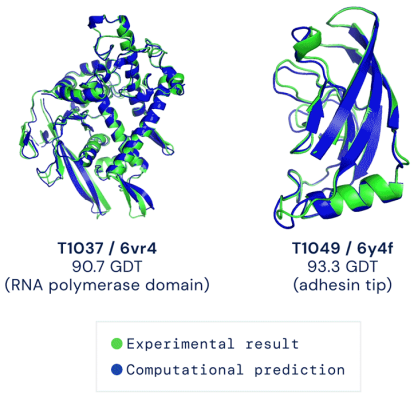
```
1  FROM pytorch/pytorch:latest
2  MAINTAINER nako.sung@navercorp.com
3
4  RUN git clone https://github.com/kakao/khaiii.git
5  WORKDIR /workspace/khaiii
6
7  RUN pip install cython
8  RUN pip install --upgrade pip
9  RUN pip install -r requirements.txt
10
11 RUN mkdir build
12 WORKDIR /workspace/khaiii/build
13
14 RUN cmake ..
15 RUN make all
16 RUN make resource
17
18 RUN apt-get update -y
19 RUN apt-get install -y language-pack-ko
20 RUN locale-gen en_US.UTF-8
21 RUN update-locale LANG=en_US.UTF-8
```

- AlphaFold[2]: a solution to a 50-year-old grand challenge in biology



DeepMind / AlphaFold

[2] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* (2021). (Accelerated article preview). DOI: 10.1038/s41586-021-03819-2.



First time setup

The following steps are required in order to run AlphaFold:

1. **Install Docker.**
   - Install NVIDIA Container Toolkit for GPU support.
   - Setup running Docker as a non-root user.

(nature.com) 'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures

4.1.3 Reproducible Data Science

(Development environments) sharing the same data science environment
(Operational environments) deploying the same data science environment to a production server

- Docker for Data Science

Docker is a tool that simplifies the installation process for software engineers. Coming from a statistics background I used to care very little about how to install software and would occasionally spend a few days trying to resolve system configuration issues. Enter the god-send Docker almighty.

Think of Docker as a light virtual machine (I apologise to the Docker gurus for using that term). Generally someone writes a **Dockerfile** that builds a **Docker Image** which contains most of the tools and libraries that you need for a project. You can use this as a base and add any other dependencies that are required for your project. Its **underlying philosophy is that if it works on my machine it will work on yours**.

- How to use Docker for your data science projects

Why do people use Docker for data science?

The complex software we use in data science can take some time and effort to set up and things don't always work consistently when software versions are slightly different. Python virtual environments and using `pip` to install the packages listed in your `requirements.txt` can help, but it's not perfect. **Using Docker means you can give everyone in your team the same data science environment, so the software runs correctly and identically on each machine, and you can also deploy your Docker container to a production server, so that works perfectly too**. Your projects will be shareable and reproducible.

- Docker frequently asked questions (FAQ)

Docker defines an abstraction for these machine-specific settings. The **exact same Docker container** can run - unchanged - on many different machines, with many different configurations.

4.1.4 Use Cases for Data Science

- Docker for Data Science – A Step by Step Guide

Examples of data science oriented docker containers

- pytorch/pytorch - a simple container for Use Case 1 that includes **Pytorch**
- jupyter/scipy-notebook - A container for Use Case 2 that includes **Jupyter as the UI**, and **many python data science modules**.

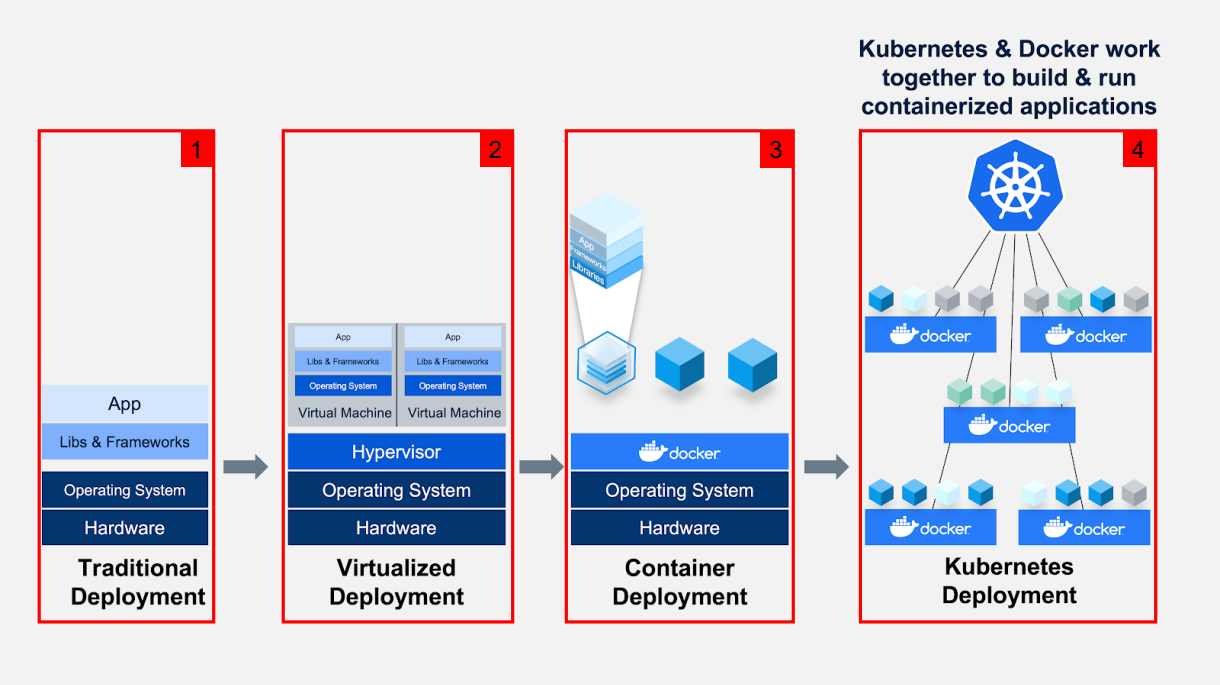
4.2 Understanding Docker Concepts and Terminology for using Docker and Docker Compose

4.2.1 Docker

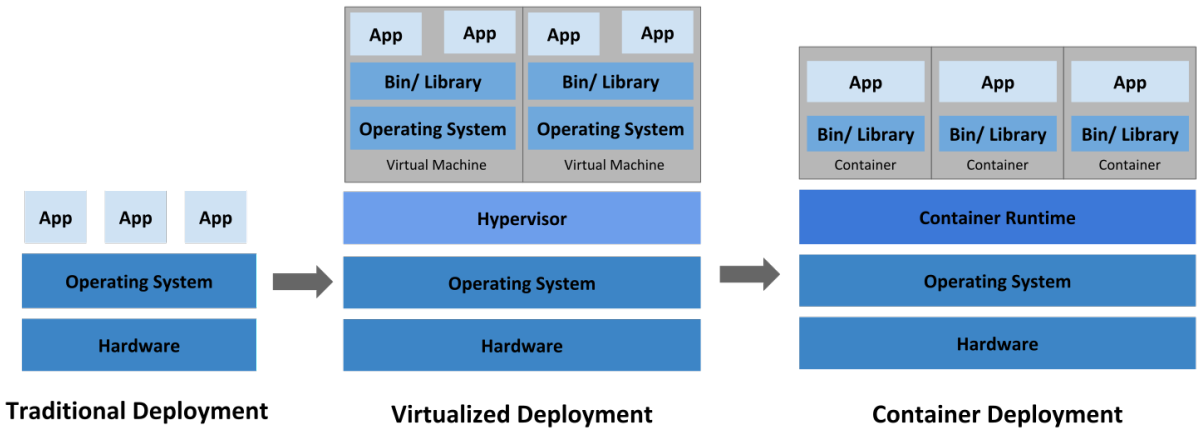
- Virtualization, Containerization, and Orchestration
- Docker Container Components

Virtualization, Containerization, and Orchestration

- Top Questions Answered: Docker and Kubernetes? I Thought You Were Competitors!



- What is Kubernetes?



Traditional deployment era: Early on, organizations ran applications on physical servers. There was no way to define **resource boundaries for applications** in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expen-

sive for organizations to maintain many physical servers.

Virtualized deployment era: As a solution, **virtualization** was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

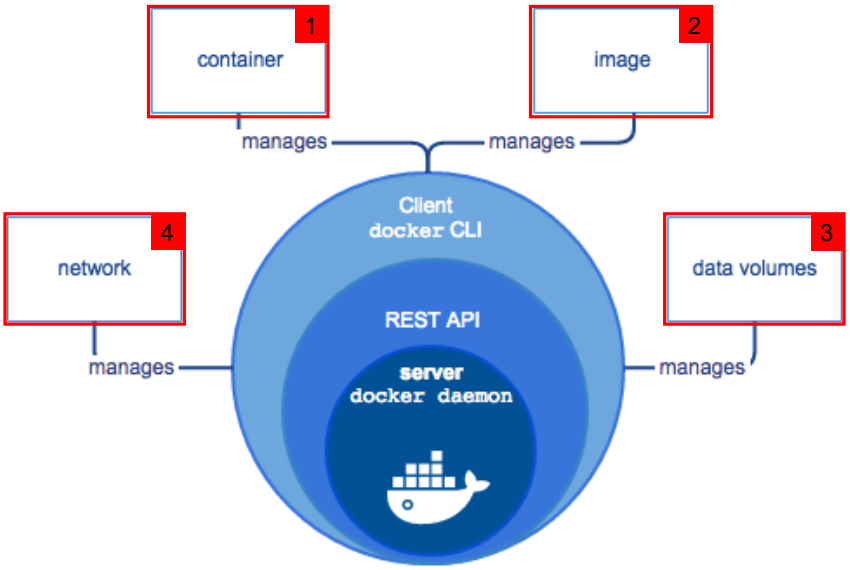
Virtualization allows **better utilization of resources** in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

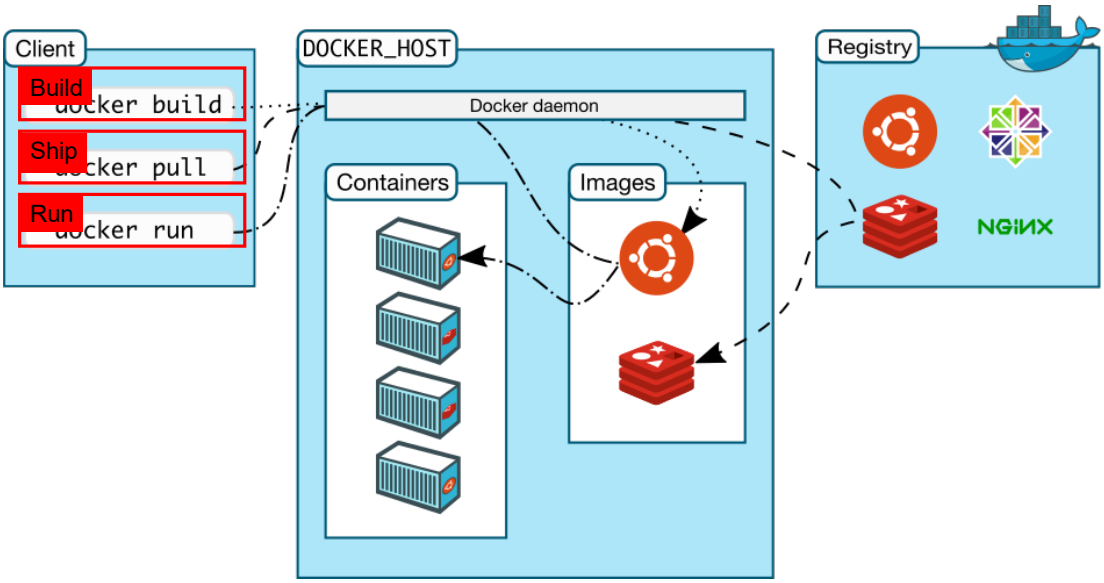
Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment era: Containers are **similar to VMs**, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered **lightweight**. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Docker Container Components

- Docker overview

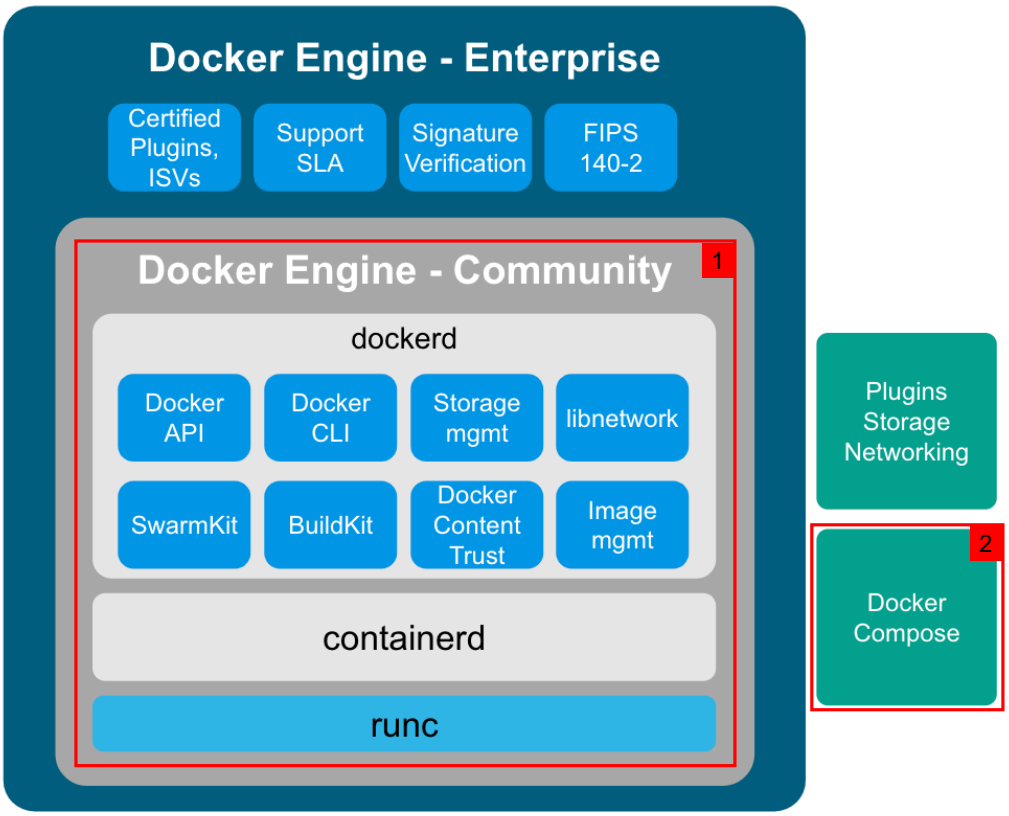




Docker architecture

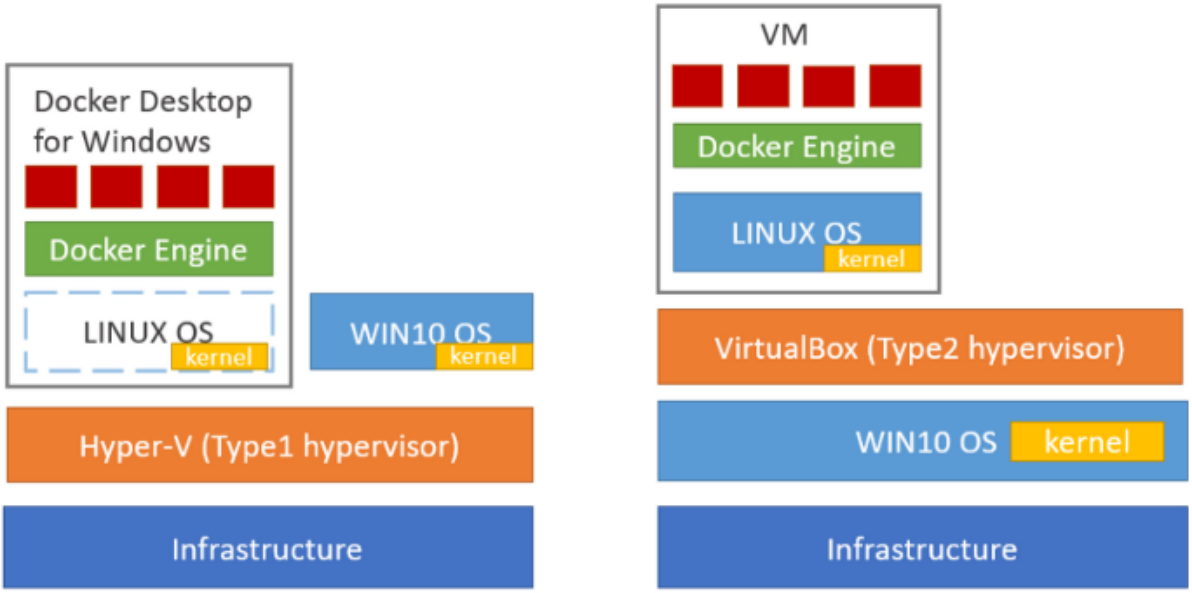
Docker uses a **client-server architecture**. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is **Docker Compose**, that lets you work with **applications consisting of a set of containers**.

- Introducing Docker Engine 18.09



4.2.2 Docker on Windows Subsystem for Linux (WSL)

- Your First Step to use Docker on a Non-Linux OS



Hypervisor: a virtual machine engine

A **hypervisor (or virtual machine monitor, VMM, virtualizer)** is **computer software, firmware, or hardware that creates and runs virtual machines**. A virtual machine is the virtualization/emulation of a computer system. For example, if you are using Win10 on your PC, a hypervisor can create a Linux virtual machine that makes you feel like using a Linux system.

Note there are **two types of hypervisors**:

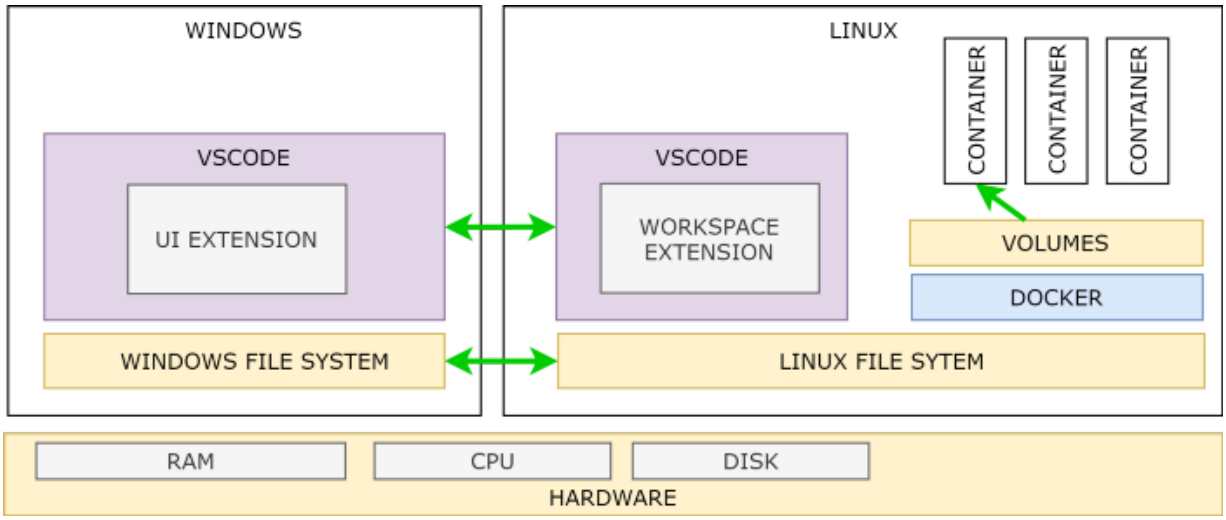
- **Type1 (Hyper-V) :** hypervisor sits on top of the hardware.
- **Type2 (Oracle VirtualBox):** hypervisor sits on top of the Host OS.

Docker Engine on Windows

- **Solution #1: Docker Desktop for Windows**
- **Solution #2: Docker on Linux VM**
- **Solution #3: Docker Toolbox (deprecated)**

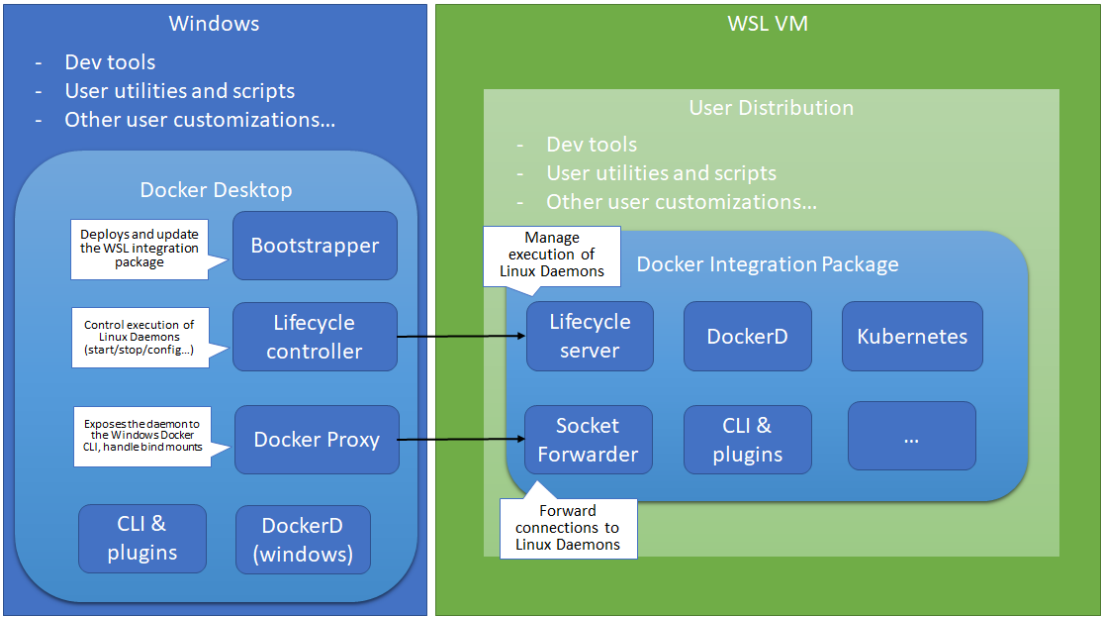
- How to Boost Docker with WSL2

  How Docker and VS Code work on WSL2



- Using Docker in WSL 2

  How it works



4.3 Setting up Docker Desktop for Windows and Using the GitLab Container Registry

4.3.1 Setting up Docker Desktop for Windows

Overview of Docker remote development on Windows

(1) Enabling a Hardware Virtualization

(2) Setting up Hyper-V

(3) **Installing Docker Desktop for Windows**

(4) **Starting Docker Desktop for Windows**

(5) **Configuring Docker Desktop for Windows**

(1) Enabling a Hardware Virtualization

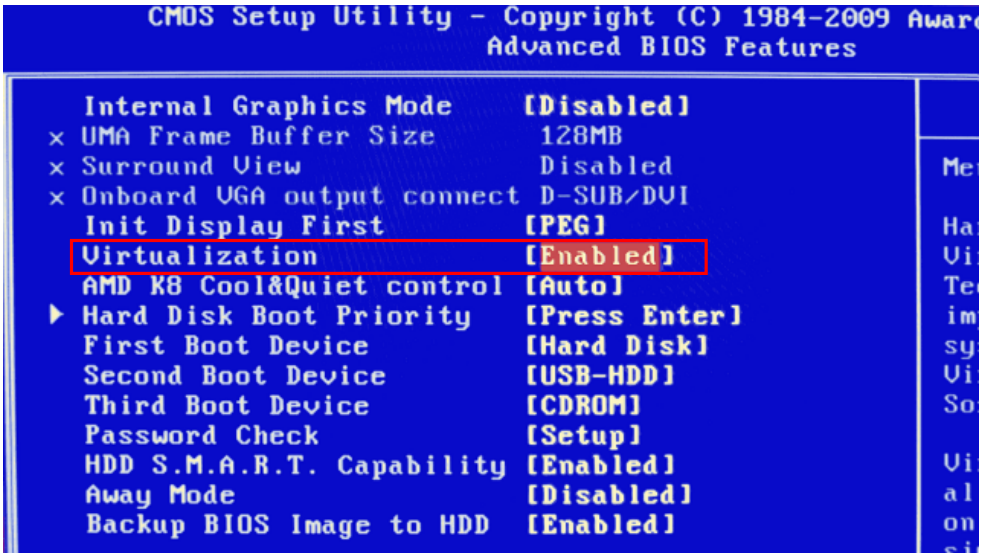
Enabling **Intel VT and AMD-V** Virtualization Hardware Extensions in BIOS

- Intel Virtual Technology (VT)
  How To Enable, Configure and Use Hyper-V on Windows 10

  > To get your computer prepared to run Hyper-V, make sure that the hardware virtualization support is enabled in the BIOS first. Boot into BIOS on your computer, enable **Virtualization Technology** under System Security. Depending on the version of BIOS you are running, you may need to poke around to find it.

- AMD Virtualization
  How do I enable hardware virtualization technology (VT-x) for use in Virtualbox?

  > Enter the BIOS (often pressing `Del` or `F12` while booting) and see with the manual how it is named there. Each BIOS appears to have a different name fror this. Search for **Virtualization, Virtualization Technology (VT-x), SVM, VMX**, or similar, here shown for an Award BIOS:



(2) Setting up Hyper-V

Install Hyper-V on Windows 10

> The Hyper-V role **cannot be installed** on **Windows 10 Home**.

Open **Command Prompt as Administrator** and run:

- **TODO:** execute `4-docker/setups/windows/hyper-v/enable-hyper-v.bat`

```
powershell dism.exe /online /enable-feature /featurename:Microsoft-Hyper-V /all /
    norestart
```

(3) Installing Docker Desktop for Windows

- **TODO:** execute `4-docker/setups/windows/docker/docker-apps.bat`

```
@echo off

echo ################################################################################
echo Installing Docker
echo ################################################################################

:: ================================================================================
:: [Deprecated] Docker CLI
:: ================================================================================
:: https://chocolatey.org/packages/docker

:: choco install -y docker
:: choco install -y docker-compose
:: choco install -y docker-machine

:: ================================================================================
:: [Deprecated] Docker Toolbox
:: ================================================================================
:: https://chocolatey.org/packages/docker-toolbox

:: choco install -y docker-toolbox

:: ================================================================================
:: [Deprecated] Docker for Windows
```

```
:: ============================================================================
:: https://chocolatey.org/packages/docker-for-windows

:: choco install -y docker-for-windows --version 18.06.1.19507

echo ============================================================================
echo Docker Desktop for Windows
echo ============================================================================
:: https://chocolatey.org/packages/docker-desktop/

:: choco install -y docker-desktop --version 2.0.0.0
:: choco install -y docker-desktop --version 2.1.0.1
:: choco install -y docker-desktop --version 2.1.0.5
:: choco install -y docker-desktop --version 2.2.0.5
:: choco install -y docker-desktop

:: wget "https://download.docker.com/win/stable/Docker Desktop Installer.exe" -P %
    USERPROFILE%\Downloads\
wget "https://desktop.docker.com/win/stable/amd64/Docker Desktop Installer.exe" -P %
    USERPROFILE%\Downloads\
%USERPROFILE%\Downloads\"Docker Desktop Installer.exe"

:: Docker Quick Start (Login Required)
:: https://hub.docker.com/?overlay=onboarding&step=download

:: TODO: Configuring Docker Desktop for Windows (Docker Desktop 2.1.x.x)
:: ----------------------------------------------------------------------------
:: 1 Expose daemon on tcp://localhost:2375 without TLS
::   Settings | General | [X] Expose daemon on tcp://localhost:2375 without TLS
:: 2 Select the local drives you want to be available to your containers: C
::   Settings | Resources | Select the local drives you want to be available to your
    containers: [X] C
```

- Install Docker Desktop on Windows

1. Double-click **Docker Desktop Installer.exe** to run the installer.

   If you haven't already downloaded the installer (`Docker Desktop Installer.exe`), you can get it from Docker Hub. It typically downloads to your `Downloads` folder, or you can run it from the recent downloads bar at the bottom of your web browser.

2. When prompted, ensure the **Enable Hyper-V Windows Features** or the **Install required Windows components for WSL 2** option is selected on the Configuration page.

3. Follow the instructions on the installation wizard to authorize the installer and proceed with the install.

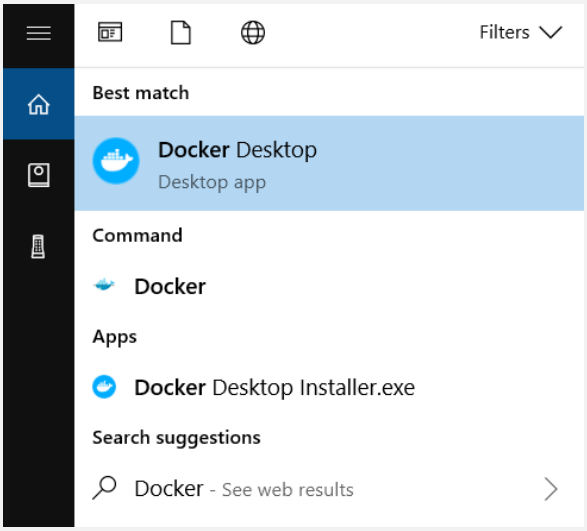
4. When the installation is successful, click **Close** to complete the installation process.

5. If your admin account is different to your user account, you must add the user to the **docker-users** group. Run **Computer Management** as an administrator and navigate to **Local Users and Groups > Groups > docker-users**. Right-click to add the user to the group. Log out and log back in for the changes to take effect.

(4) Starting Docker Desktop for Windows

- Install Docker Desktop on Windows

Docker Desktop does not start automatically after installation. To start Docker Desktop, search for Docker, and select **Docker Desktop** in the search results.



When the **whale icon** in the **status bar** stays steady, Docker Desktop is up-and-running, and is accessible from any terminal window.

If the **whale icon** is hidden in the **Notifications area**, click the up arrow on the taskbar to show it. To learn more, see Docker Settings.

(5) Configuring Docker Desktop for Windows

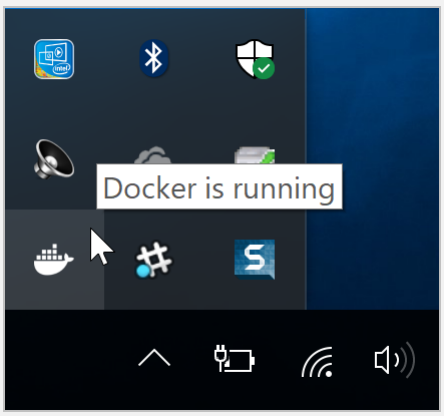
- Docker Desktop for Windows user manual

Settings

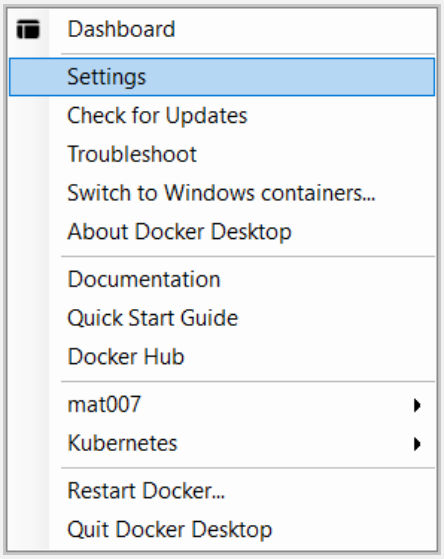
The **Docker Desktop** menu allows you to open the Docker Dashboard, run the Quick Start Guide, configure your Docker settings such as installation, updates, version channels, Docker Hub login, and more.

This section explains the configuration options accessible from the **Settings** dialog.

1. To open the Docker Desktop menu, right-click the Docker icon in the Notifications area (or System tray):

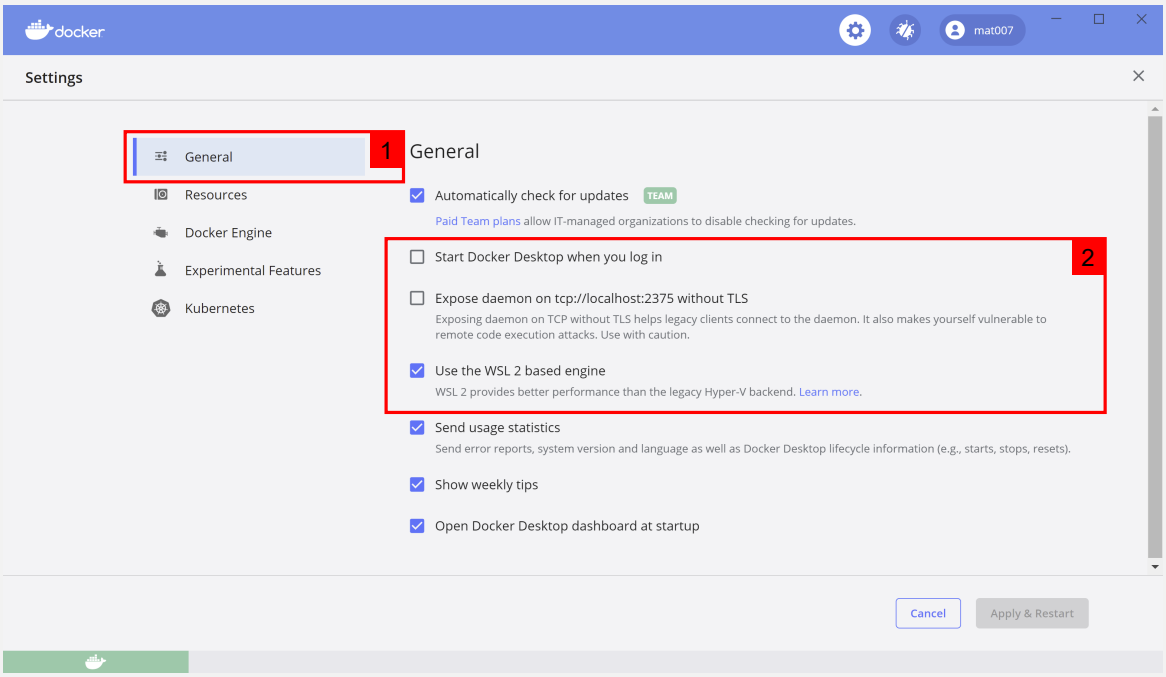


2. Select **Settings** to open the Settings dialog:



General

On the General tab of the Settings dialog, you can configure when to start and update Docker.



- **Start Docker when you log in:** Select this option to automatically start Docker Desktop when you log into your Windows machine.
- **Expose daemon on tcp://localhost:2375 without TLS:** Click this option to **enable legacy clients to connect to the Docker daemon**. You must use this option with caution as exposing the daemon without TLS can result in **remote code execution attacks**.
- **Use the WSL 2 based engine:** WSL 2 provides better performance than the legacy Hyper-V backend. For more information, see Docker Desktop WSL 2 backend.

- Docker Desktop WSL 2 backend

When Docker Desktop restarts, go to **Settings > Resources > WSL Integration**.

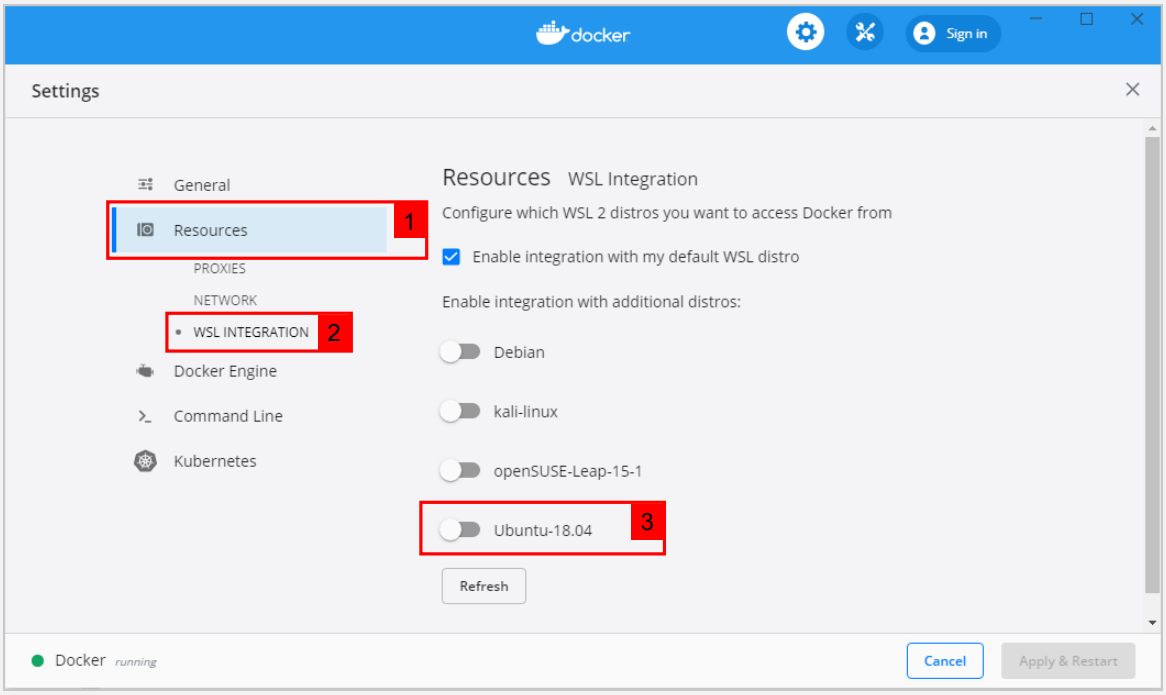
The Docker-WSL integration will be enabled on your default WSL distribution. To change your default WSL distro, run `wsl --set-default <distro name>`.

For example, to set Ubuntu as your default WSL distro, run `wsl --set-default ubuntu`.

Optionally, select any additional distributions you would like to enable the Docker-WSL integration on.

Note

The **Docker-WSL integration components** running in your distro depend on **glibc**. This can cause issues when running **musl-based distros such as Alpine Linux**. Alpine users can use the alpine-pkg-glibc package to deploy glibc alongside musl to run the integration.



4.3.2 Using the GitLab Container Registry

- GitLab Container Registry

With the **Docker Container Registry** integrated into GitLab, **every GitLab project can have its own space to store its Docker images**.

You can read more about Docker Registry at https://docs.docker.com/registry/introduction/.

This document is the user guide. To learn how to enable the Container Registry for your GitLab instance, visit the administrator documentation.

Viewing the GitLab Container Registry

- View the Container Registry

You can view the Container Registry for a project or group.

1. Go to your project or group.
2. Go to **Packages & Registries > Container Registry**.

You can search, sort, filter, and delete containers on this page. You can share a filtered view by copying the URL from your browser.

Only members of the project or group can access a private project's Container Registry.

If a project is public, so is the Container Registry.

Creating a personal access token

- Creating a personal access token

You can create as many personal access tokens as you like from your GitLab profile.

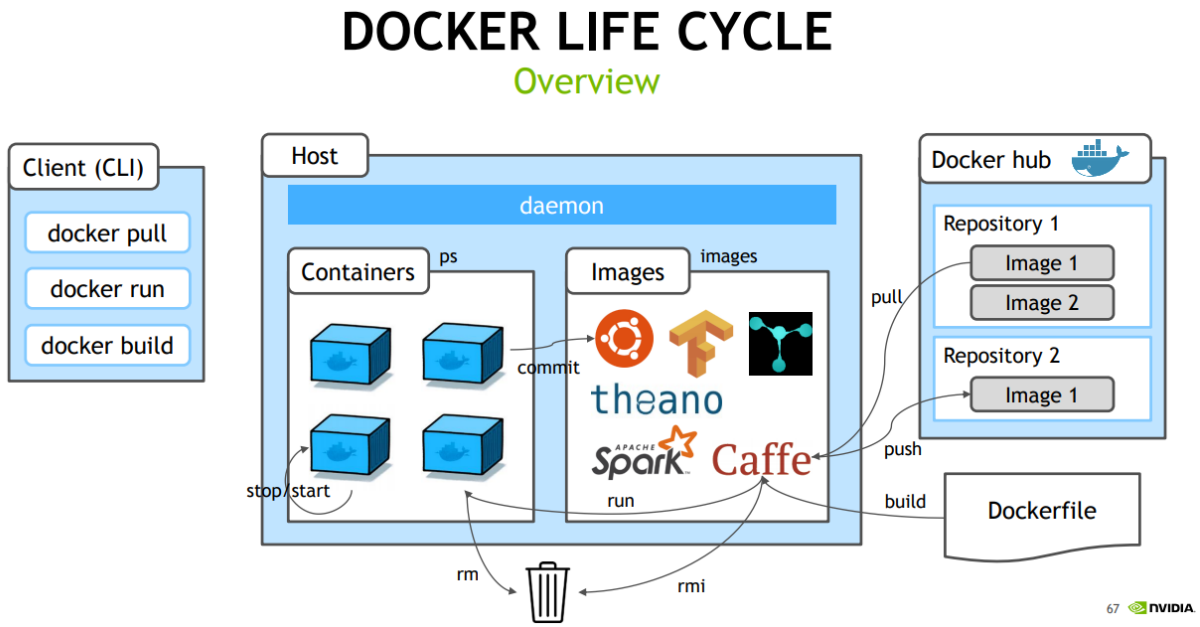
1. Log in to GitLab.
2. In the upper-right corner, click your avatar and select **Settings (⟶ Preferences)**.
3. On the **User Settings** menu, select **Access Tokens**.
4. Choose a name and optional expiry date for the token.
    - Token name: **Docker Image Registry**
5. Choose the desired scopes.
    - Select scopes: {**select all options**}
6. Click the **Create personal access token** button.
7. Save the personal access token somewhere safe. Once you leave or refresh the page, you won't be able to access it again.

4.4 Using Docker and Docker Compose

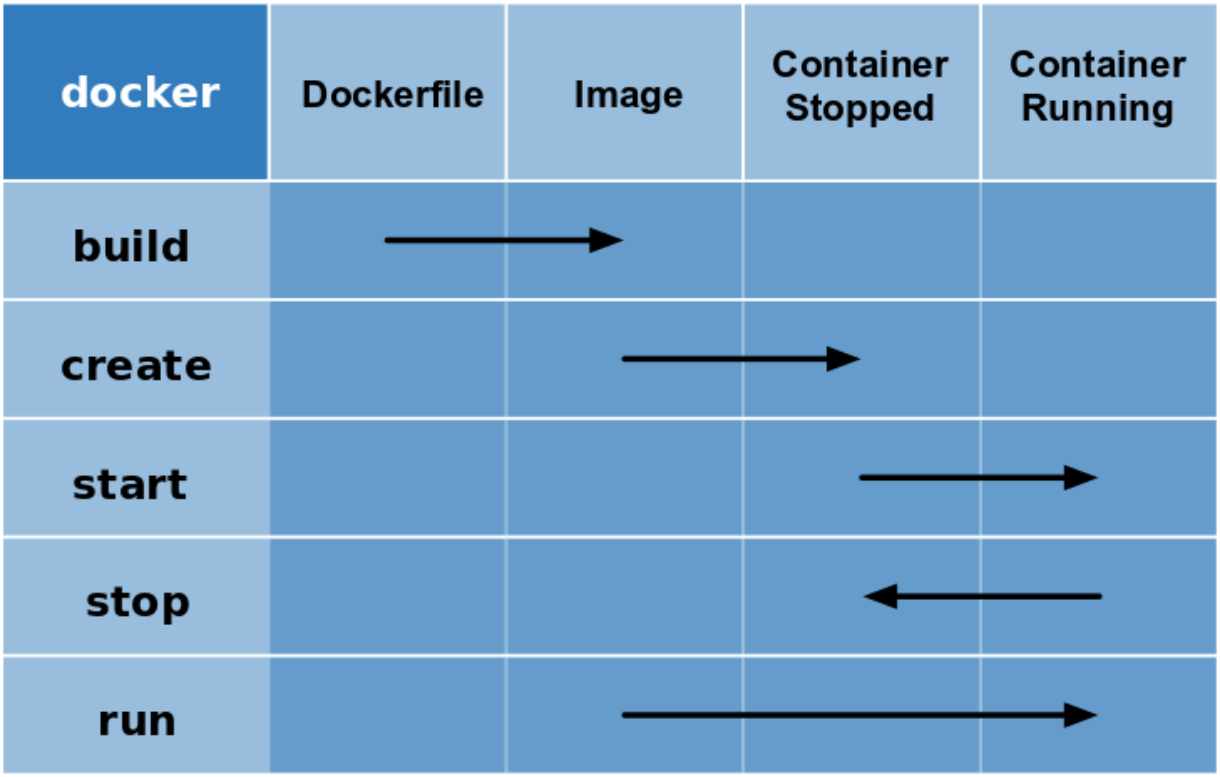
4.4.1 Basice Commands for Docker and Docker Compose

Docker Cheat Sheets and Commands

- DGX-1 DOCKER USER GUIDE 17.08



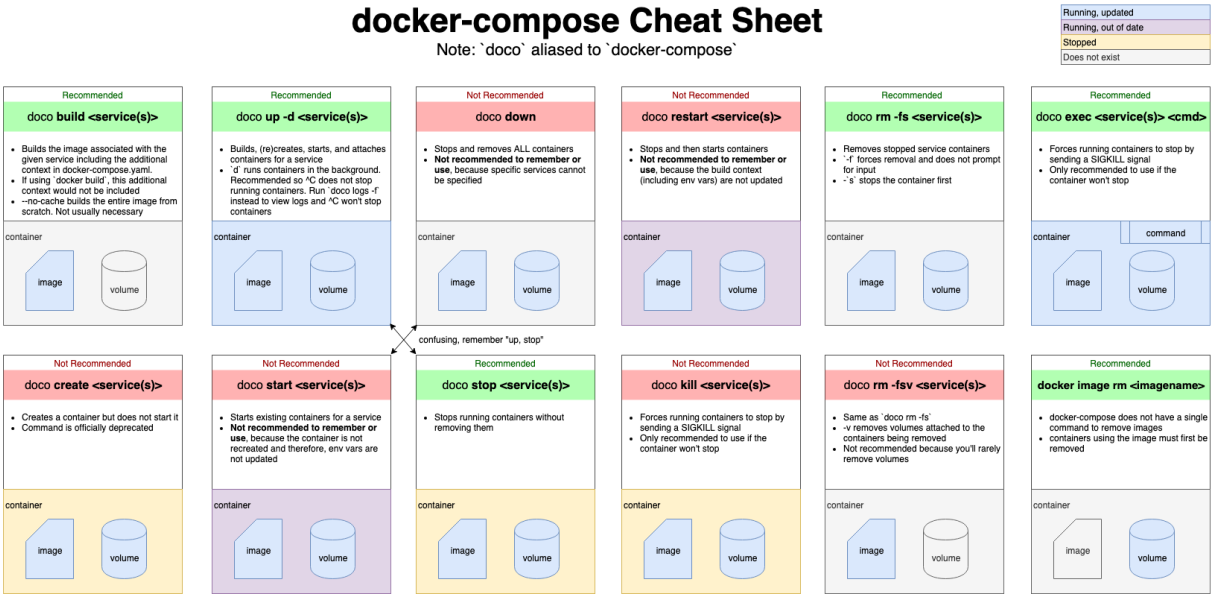
- Transformations of the most basic commands



Command	Description	Usage	Important Options	Example
`build`	Create a new **image** from a Dockerfile.	`[OPTIONS] DIR`	`-t` Define image name and tag	`docker build -t myapp:0.0.1 /home/user/app` Create a new image called myapp with tag 0.0.1 from the Dockerfile in directory /home/user/app. The new image is now listed in `docker images`.
`create`	Create a new **container** from an image.	`[OPTIONS] IMAGE [CMD] [CMDARG…]`	`-e` Set environment variables `-i` Interactive (allow to attach local stdin, stdout, and stderr) `-p` Define port mapping `-t` Set up a terminal in the container `-v` Define a volume	`docker create -i -t -e FOO=bar -p 8080:80 -v /home/user:/root ubuntu:16.04 bash` Create a new **Ubuntu 16.04** container with an **interactive terminal** running **Bash**. Set the environment variable **FOO** in the container. Expose **port 80** of the container, an map it to port **8080** on the host. Map the **/home/user** directory on the host to **/root** in the container. The new container is now listed in `docker ps -a`.
`start`	Start a container.	`[OPTIONS] CONTAINER…`	`-a` Attach **stdout** and **stderr** (show stdout and stderr of container on host) `-i` Attach **stdin** (read stdin for container from host)	`docker start -ai f585a987fae2` Start a previously created container and attach local stdout, stderr, and stdin to it. The `-a` and `-i` options together have the same effect as running `docker attach` afterwards. The container is now listed in `docker ps`.
`attach`	Attach local **stdin**, **stdout**, and **stderr** to running **container**.	`[OPTIONS] CONTAINER`		`docker attach f585a987fae2` Attach local stdout, stderr, and stdin to a previously started container. If the container runs a shell, this is like logging in to the container.
`stop`	Stop a running container.	`[OPTIONS] CONTAINER…`		`docker stop f585a987fae2` Stop a running container. The container is now still listed in `docker ps -a`, but not anymore in `docker ps`.
`run`	**Create** a new container from an image, **start** it, and **attach** to it.	`[OPTIONS] IMAGE [CMD] [CMDARG…]`	`-d` Don't attach to the container `-e` Set environment variables `-i` Interactive (allow to attach local stdin, stdout, and stderr) `-p` Define port mapping `-t` Set up a terminal in the container `-v` Define a volume	`docker run -i -t -e FOO=bar -p 8080:80 -v /home/user:/root ubuntu:16.04 bash` Same example as for `create`, but now also start the container and attach local stdin, stdout, and stderr to it. The `run` command merges the three commands `create`, `start`, and `attach` into one.
`ps`	List running containers	`[OPTIONS]`	`-a` List all containers, not just running ones `-q` List only IDs of containers	`docker ps -aq` List IDs of all existing containers.
`images`	List images.	`[OPTIONS]`	`-a` Also list intermediate images `-q` List only IDs of images	`docker images -q` List IDs of all main images.
`rm`	Remove containers.	`[OPTIONS] CONTAINER…`	`-f` Force removal `-v` Also remove volumes	`docker rm -f $(docker ps -aq)` Remove all existing containers.
`rmi`	Remove images	`[OPTIONS] IMAGE…`	`-f` Force removal	`docker rmi -f $(docker images -q)` Remove all existing images.

Docker Compose Cheat Sheets and Commands

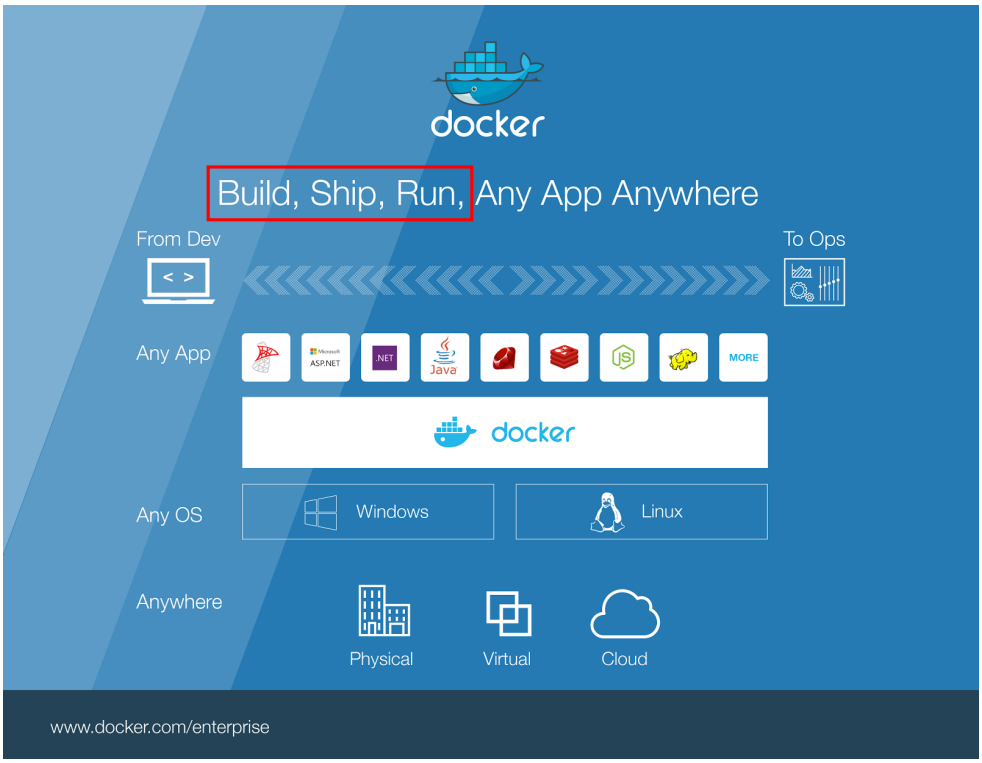
- docker-compose cheat sheet



4.4.2 Using Docker and Docker Compose with Commands

Using Docker with Commands

- Introducing Docker for Windows Server 2016




```
# Docker environment information
docker version
docker info

# Building Docker images from Dockerfile
docker images
docker image ls
docker image ls -a
docker build -t registry.gitlab.com/siaiedu/python-on-linux:latest .
docker image build -t registry.gitlab.com/siaiedu/python-on-linux:latest .
docker rmi registry.gitlab.com/siaiedu/python-on-linux:latest
docker image rmi registry.gitlab.com/siaiedu/python-on-linux:latest
docker rmi -f 'docker images -aq'
docker image prune -a

# Shipping; sharing Docker images
docker login registry.gitlab.com
docker pull registry.gitlab.com/siaiedu/python-on-linux:latest
docker image pull registry.gitlab.com/siaiedu/python-on-linux:latest
docker push registry.gitlab.com/siaiedu/python-on-linux:latest
docker image push registry.gitlab.com/siaiedu/python-on-linux:latest

# Running Docker containers
docker ps
docker ps -a
docker container ls
docker container ls -a
docker run --rm -it registry.gitlab.com/siaiedu/python-on-linux:latest
docker container run --rm -it registry.gitlab.com/siaiedu/python-on-linux:latest
docker run -d --name python-on-linux registry.gitlab.com/siaiedu/python-on-linux:
    latest
docker container run -d --name python-on-linux registry.gitlab.com/siaiedu/python-on
    -linux:latest
docker exec -it python-on-linux bash
docker container exec -it python-on-linux bash
docker stop python-on-linux
docker container stop python-on-linux
```

```
docker rm python-on-linux
docker container rm python-on-linux

# volume commands
docker volume ls

# network commands
docker network ls
```

- (Docker Hub) Hello World! (an example of minimal Dockerization)

```
docker pull hello-world
docker images hello-world
docker run hello-world

# https://hub.docker.com/_/ubuntu
docker run -it ubuntu bash

# https://hub.docker.com/r/tensorflow/tensorflow
docker run -it --rm tensorflow/tensorflow bash
```

Using Docker Compose with Commands

- docker compose

`docker-compose.yml`

```
services:
  webapp:
    image: examples/web
    ports:
      - "8000:8000"
    volumes:
      - "/data"
```

`docker-compose.admin.yml`

```
services:
  webapp:
    build: .
    environment:
      - DEBUG=1
```

```
# docker compose with docker-compose.yml
docker-compose -f docker-compose.yml -f docker-compose.admin.yml run backup_db
docker compose -f docker-compose.yml -f docker-compose.admin.yml run backup_db

docker compose ps
docker compose up -d
docker compose down
docker compose down --rmi all
```

- Quickstart: Compose and Django

`docker-compose.yml`

```
version: "3.9"

services:
  db:
    image: postgres
    volumes:
```

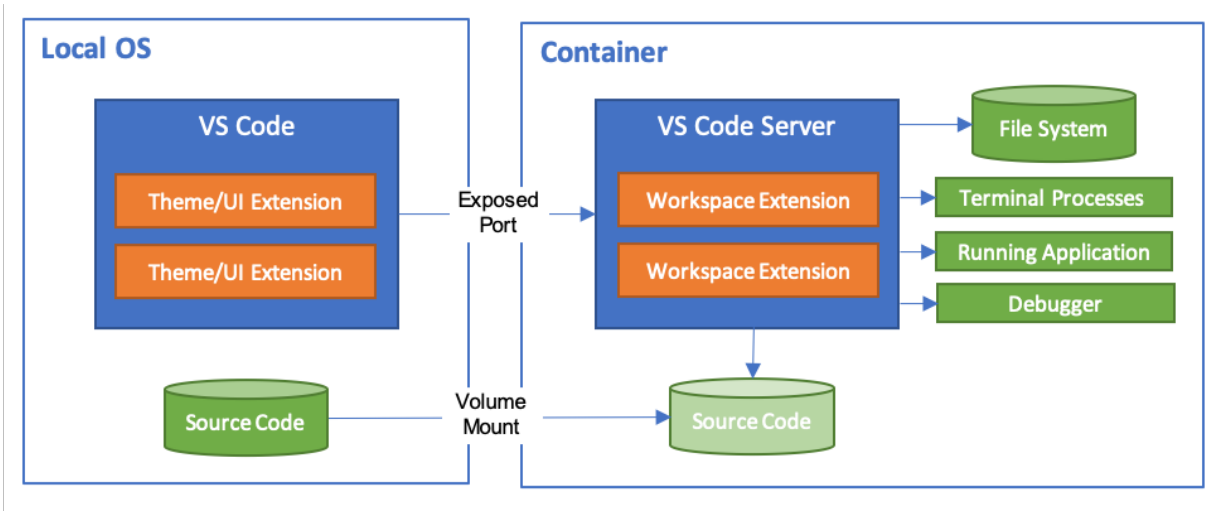
```
      - ./data/db:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
```

```
# docker compose with docker-compose.yml
docker-compose run web django-admin startproject composeexample .
```

4.4.3 Using Docker and Docker Compose in VS Code for Python Development

Understanding Local and Container Environments for Developing inside a Container

- Developing inside a Container



Installing VS Code extensions for Docker

- Docker for Visual Studio Code

> The **Docker extension** makes it easy to **build, manage, and deploy containerized applications from Visual Studio Code**. It also provides one-click debugging of Node.js, Python, and .NET Core inside a container.

- Remote - Containers

> The extension starts (or attaches to) a development container running a well defined tool and run-time stack. **Workspace files can be mounted into the container from the local file system, or copied or cloned into it once the container is running**. Extensions are installed and run inside the container where they have full access to the tools, platform, and file system.

Creating a Simple Python Application using Docker

(1) Building a Python image

- Build your Python image, Python in a container

`4-docker/Dockerfile`

```
# syntax=docker/dockerfile:1

# python Docker Official Images
# https://hub.docker.com/_/python/
FROM python:3.9-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

EXPOSE 5000

CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0", "--port=5000"]
```

`4-docker/app.py`

```
from flask import Flask

app = Flask(__name__)


@app.route("/")
def hello_world():
    return "Hello, Docker!"
```

```
docker build --pull --rm -f Dockerfile -t python-docker:latest .
```

```
Dockerfile
> {mouse right-click} > Build Image...
> Tag image as... > {tag name}
```

  - tag name: **python-docker:latest**

- Create a DockerFile related to an existing image

you can see the steps to add the docker files to a project by using the Docker Extension for VS Code:

1. Open the command palette, type ”**docker**” and select ”**Add Docker Files to Workspace**”.
2. Select Application Platform (ASP.NET Core)
3. Select Operating System (Linux)
4. Include optional Docker Compose files
5. Enter ports to publish (80, 443)
6. Select the project

- Docker, Kaniko, Buildah
  Different ways to build container images

(2) Running the Python image as a Container

- Run your image as a container

```
docker run --rm -d python-docker:latest
```

```
Docker
> IMAGES > {image repository name} > {image tag name} > {mouse right-click}
> Run
```

  - image repository name: **python-docker**
  - image tag name: **latest**
- Visit http://localhost:5000/

(3) Connecting to the Docker container using Remote-Containers

- Attach to a running container

> VS Code supports **image or container name-level configuration files** to speed up setup when you repeatedly connect to a given Docker container. Once attached, anytime you open a folder, install an extension, or forward a port, **a local image-specific configuration file will automatically be updated to remember your settings** so that when you attach again, everything is back to the right place.

```
docker exec -it {container name} bash
```

```
Docker
> CONTAINERS
> {the container to attach to} > {mouse right-click}
> Attach Visual Studio Code
```

```
The Command Palette (Ctrl+Shift+P or F1)
> Remote-Containers: Attach to Running Container...
> Select the container to attach VS Code > {container name}
```

- Create a development container

> The **Visual Studio Code Remote - Containers** extension lets you use a Docker container as a full-featured development environment. It allows you to open any folder or repository inside a container and take advantage of Visual Studio Code's full feature set. A `devcontainer.json` file in your project tells VS Code how to access (or create) a **development container** with a well-defined tool and runtime stack. This container can be used to run an application or to separate tools, libraries, or runtimes needed for working with a codebase.

(4) Stopping and Removing the Docker container

-

```
Docker
> CONTAINERS > Individual Containers
> {select the container to attach to} > {mouse right-click}
> Stop
```

```
Docker
> CONTAINERS > Individual Containers
> {select the container to attach to} > {mouse right-click}
> Remove...
```

(5) Removing the Docker image

-

```
Docker
> IMAGES > {image repository name} > {image tag name} > {mouse right-click}
> Remove...
```

Creating a Python Application using Docker and Docker Compose for Data Science

(1) **Creating a Dockerfile**

`4-docker/docker/Dockerfile`

```
1  # syntax=docker/dockerfile:1
2
3  ##############################################################################
4  # Dockerfile for Python on Linux
5  ##############################################################################
6
7  # python Docker Official Images
8  # https://hub.docker.com/_/python/
9
10 # ============================================================================
11 # Set base image and environment variables
12 # ============================================================================
13
14 ARG PYTHON_VERSION=3.9-slim-buster
```

```

# Full official Debian-based Python image
FROM python:${PYTHON_VERSION}

# Sets utf-8 encoding for Python et al
ENV LANG=C.UTF-8
# Turns off writing .pyc files; superfluous on an ephemeral container.
ENV PYTHONDONTWRITEBYTECODE=1
# Seems to speed things up
ENV PYTHONUNBUFFERED=1

# Always set a working directory
WORKDIR /app

# ================================================================================
# Installing application dependencies
# ================================================================================

# Install Python dependencies
COPY ./requirements /requirements
RUN pip install -U pip \
    && pip install --no-cache-dir -r /requirements/dev.txt

# ================================================================================
# Copying the src code
# ================================================================================

# Adding source code
COPY ./src .

# ================================================================================
# Running jupyter notebook
# ================================================================================

EXPOSE 8888

# --NotebookApp.token ='demo' is the password
CMD ["jupyter", "notebook", "--no-browser", "--ip=0.0.0.0", "--allow-root", "--
    NotebookApp.token='siai'"]
```

(2) **Creating a Docker Compose file**

`4-docker/docker/docker-compose.all.yml`

```
version: '2.2'

services:

  app:
    container_name: python-on-linux
    restart: always
    build:
      context: ..
      dockerfile: ./docker/Dockerfile
    env_file:
      - ../environment.env
    environment:
      - ENV_CONFIG=development
    # for local development
    volumes:
      - ../:/app
    depends_on:
      db:
        condition: service_started
```

```
    ports:
      - "8888:8888"
    networks:
      - service

  # postgres
  # https://hub.docker.com/_/postgres
  db:
    container_name: db
    restart: always
    image: postgres:9.6
    environment:
      - POSTGRES_USER=siai
      - POSTGRES_PASSWORD=siai
      - POSTGRES_DB=siai
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - service
    ports:
      - "5432:5432"

  # Container Deployment
  # https://www.pgadmin.org/docs/pgadmin4/4.15/container_deployment.html
  db_admin:
    container_name: db_admin
    restart: always
    image: dpage/pgadmin4
    depends_on:
      db:
        condition: service_started
    environment:
      - PGADMIN_DEFAULT_EMAIL=dev@siai.org
      - PGADMIN_DEFAULT_PASSWORD=siai
    networks:
      - service
    ports:
      - "8000:80"

volumes:
  db_data: {}

networks:
  service:
    driver: bridge
```

(3) **Up and running the Docker containers**

```
{Docker Compose YAML file}
> {mouse right-click} > Compose Up
```

- Docker Compose YAML file: **docker-compose.all.yml**

(4) **Visiting web pages**

- Jupyter Notebook: http://localhost:8888/
  - Password or token: **siai**
- pgAdmin: http://localhost:8000/
  - Email Address / Username: **dev@siai.org**

- Password: **siai**
- Add New Server > Host name/address:
  {**IPAddress** of the postgres container by `Inspect` command}

(5) **Attaching to the Docker container using Remote-Containers**

```
Docker
> CONTAINERS
> {the container to attach to} > {mouse right-click}
> Attach Visual Studio Code
```

- the container to attach to: **python-on-linux**

(6) **Setting up Python development enviornment and Running Python scripts**

- Open the directory `/app`
- Install VS Code extensions: Python
- Select Python Interpreter: `/usr/local/bin/python`
- Run the Python scripts: `/app/src/hello.py`, `/app/src/iris_kmeans_dash.py`

(7) **Pushing the Docker Image to GitLab Image Registry**

- Build and push images by using Docker commands

```
1 docker login registry.gitlab.com
2 docker login registry.gitlab.com -u <username> -p <token>
3
4 docker build -t registry.gitlab.com/siaiedu/python-on-linux -f ./Dockerfile .
5
6 docker push registry.gitlab.com/siaiedu/python-on-linux
7 docker push registry.gitlab.com/siaiedu/python-on-linux/docker_app:latest
```

```
Docker
> REGISTRIES > Connect Registry...
> GitLab
> Enter your username: {username}
> GitLab Personal Access Token: {Personal Access Token}
```

- username: **siaiedu**

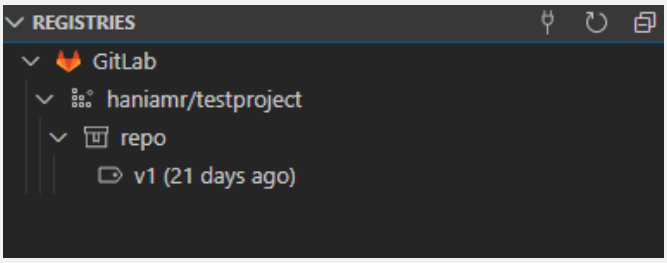
```
Docker
> IMAGES > {image repository name} > {image tag name} > {mouse right-click}
> Push...
> Select project > {project name}
> Tag image as.. > {image name}
```

- project name: **siaiedu/python-on-linux**
- image name: **registry.gitlab.com/siaiedu/python-on-linux/docker_app:latest**

```
Docker
> REGISTRIES > Refresh
```

- Using container registries > GitLab

This connects to **Docker registries in your GitLab account**. Once you select this option, you will be required to type in your GitLab account credentials.




```
Docker
> REGISTRIES > GitLab > {project name} > {image name} > {mouse right-click}
> Pull Image
```

4.4.4 Using Docker Containers for Data Science

Anaconda Individual Edition

- Anaconda Individual Edition

Open Source

Anaconda Individual Edition is the world's most popular Python distribution platform with over 25 million users worldwide. You can trust in our long-term commitment to supporting the **Anaconda open-source ecosystem**, the platform of choice for Python data science.

Build machine learning models

Build and train machine learning models using the best Python packages built by the open-source community, including scikit-learn, TensorFlow, and PyTorch.

- (hub.docker.com) continuumio/anaconda3

docker-anaconda

Docker container with a bootstrapped installation of Anaconda (based on Python 3.X) that is ready to use.

The Anaconda distribution is installed into the `/opt/conda` folder and ensures that the default user has the `conda` command in their path.

Anaconda is the leading open data science platform powered by Python. The open source version of Anaconda is a high performance distribution and **includes over 100 of the most popular Python packages for data science**. Additionally, it provides **access to over 720 Python and R packages** that can easily be installed using the conda dependency and environment manager, which is included in Anaconda.

```
docker pull continuumio/anaconda3
docker run -i -t continuumio/anaconda3 /bin/bash
```

- (GitHub.com) Anaconda and Miniconda Docker Docker Docker

`docker-images/anaconda3/debian/Dockerfile`

```
FROM debian:buster-slim

ENV LANG=C.UTF-8 LC_ALL=C.UTF-8
ENV PATH /opt/conda/bin:$PATH

# hadolint ignore=DL3008
RUN set -x && \
    apt-get update --fix-missing && \
    apt-get install -y --no-install-recommends \
        bzip2 \
        ca-certificates \
        git \
        libglib2.0-0 \
        libsm6 \
        libxext6 \
        libxrender1 \
        mercurial \
        openssh-client \
        procps \
        subversion \
        wget \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/* && \
    UNAME_M="$(uname -m)" && \
    if [ "${UNAME_M}" = "x86_64" ]; then \
        ANACONDA_URL="https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86_64.sh"; \
        SHA256SUM="2751ab3d678ff0277ae80f9e8a74f218cfc70fe9a9cdc7bb1c137d7e47e33d53"; \
    elif [ "${UNAME_M}" = "s390x" ]; then \
        ANACONDA_URL="https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-s390x.sh"; \
        SHA256SUM="a7d1a83279f439e7d8a6c53aa725552e195c0b96ae7e7fa63baefdf0118f7942"; \
    elif [ "${UNAME_M}" = "aarch64" ]; then \
        ANACONDA_URL="https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-aarch64.sh"; \
        SHA256SUM="3a3d5a61df5422f7c8c7816217b926ec7e200cc6d62967541adead8ec46d935d"; \
    elif [ "${UNAME_M}" = "ppc64le" ]; then \
        ANACONDA_URL="https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-ppc64le.sh"; \
        SHA256SUM="097064807a9adae3f91fc4c5852cd90df2b77fc96505929bb25bf558f1eef76f"; \
    fi && \
    wget "${ANACONDA_URL}" -O anaconda.sh -q && \
```

```
    echo "${SHA256SUM} anaconda.sh" > shasum && \
    sha256sum --check --status shasum && \
    /bin/bash anaconda.sh -b -p /opt/conda && \
    rm anaconda.sh shasum && \
    ln -s /opt/conda/etc/profile.d/conda.sh /etc/profile.d/conda.sh && \
    echo ". /opt/conda/etc/profile.d/conda.sh" >> ~/.bashrc && \
    echo "conda activate base" >> ~/.bashrc && \
    find /opt/conda/ -follow -type f -name '*.a' -delete && \
    find /opt/conda/ -follow -type f -name '*.js.map' -delete && \
    /opt/conda/bin/conda clean -afy

CMD [ "/bin/bash" ]
```

RAPIDS

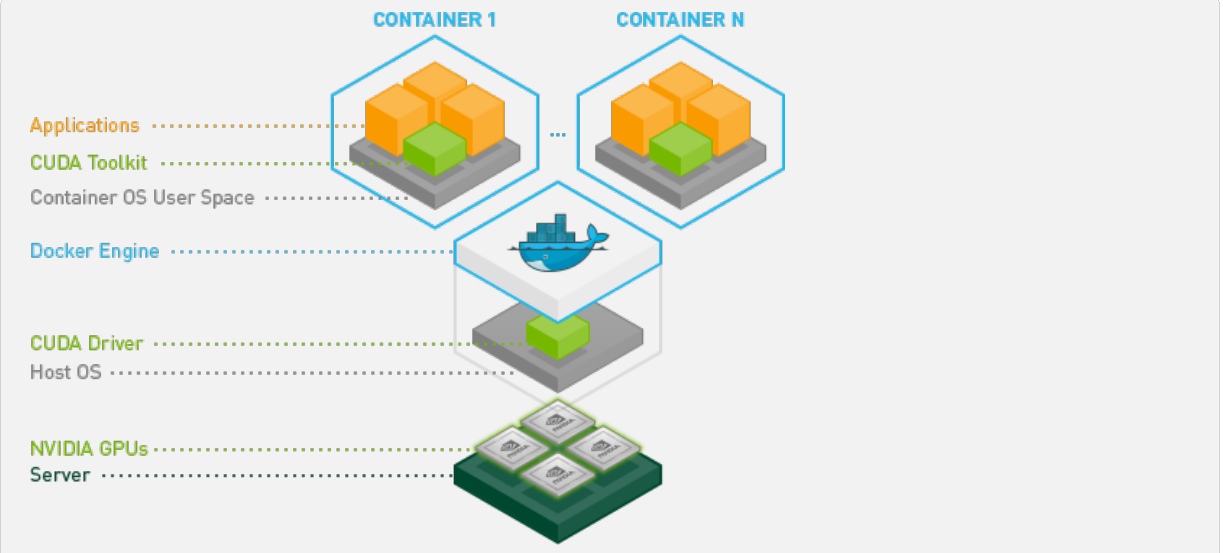
- RAPIDS > Getting Started

PREREQUISITES

- **GPU**: NVIDIA Pascal™ or better with compute capability 6.0+
- **OS**: Ubuntu 16.04/18.04/20.04 or CentOS 7/8 with gcc/++ 9.0+
  - See RDN 8 for recent changes to gcc/++ 9.0 requirements
  - RHEL 7/8 support is provided through CentOS 7/8 builds/installs
- **Docker**: Docker CE v19.03+ and **nvidia-container-toolkit**
  - Legacy Support - Docker CE v17-18 and nvidia-docker2
- **CUDA & NVIDIA Drivers**: One of the following supported versions:
  - 11.0 & v450.51+
  - 11.2 & v460.32+

```
docker pull rapidsai/rapidsai-dev:21.06-cuda11.2-devel-ubuntu20.04-py3.8
docker run --gpus all --rm -it -p 8888:8888 -p 8787:8787 -p 8786:8786 \
    rapidsai/rapidsai-dev:21.06-cuda11.2-devel-ubuntu20.04-py3.8
```

- NVIDIA Container Toolkit



Introduction

The **NVIDIA Container Toolkit** allows users to build and run **GPU accelerated Docker containers**. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPUs.

Product documentation including an architecture overview, platform support, installation and usage guides can be found in the documentation repository.

Frequently asked questions are available on the wiki.

`nvidia-docker/docker/Dockerfile.ubuntu`

```
ARG BASEIMAGE
FROM ${BASEIMAGE}

# packaging dependencies
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y --no-install-recommends \
        dh-make \
        fakeroot \
        build-essential \
        devscripts \
        lsb-release && \
    rm -rf /var/lib/apt/lists/*

# packaging
ARG PKG_VERS
ARG PKG_REV
ARG RUNTIME_VERSION
ARG DOCKER_VERSION

ENV DEBFULLNAME "NVIDIA CORPORATION"
ENV DEBEMAIL "cudatools@nvidia.com"
ENV REVISION "$PKG_VERS-$PKG_REV"
ENV DOCKER_VERSION $DOCKER_VERSION
ENV RUNTIME_VERSION $RUNTIME_VERSION
ENV SECTION ""

# output directory
ENV DIST_DIR=/tmp/nvidia-docker2-$PKG_VERS
RUN mkdir -p $DIST_DIR /dist

# nvidia-docker 2.0
COPY nvidia-docker $DIST_DIR/nvidia-docker
COPY daemon.json $DIST_DIR/daemon.json

WORKDIR $DIST_DIR
COPY debian ./debian

RUN sed -i "s;@VERSION@;${REVISION};" debian/changelog && \
    sed -i "s;@VERSION@;${PKG_VERS};" $DIST_DIR/nvidia-docker && \
    if [ "$REVISION" != "$(dpkg-parsechangelog --show-field=Version)" ]; then echo "
    $(dpkg-parsechangelog --show-field=Version)" && exit 1; fi

CMD export DISTRIB="$(lsb_release -cs)" && \
    debuild --preserve-env --dpkg-buildpackage-hook='sh debian/prepare' -i -us -uc -
    b && \
    mv /tmp/*.deb /dist
```

Lambda Stack

- Lambda Stack

Lambda Stack is all the AI software you need, and it's always up to date

Lambda Stack provides a one line installation and managed upgrade path for: **PyTorch, TensorFlow, CUDA, cuDNN, and NVIDIA Drivers**. It's compatible with Ubuntu 20.04 LTS, 18.04 LTS, and 16.04 LTS. No more futzing with your Linux AI software, Lambda Stack is here.

Everyone loves Lambda Stack — used by the F500, research labs, and the DOD

Every laptop, workstation, and server that we ship comes pre-installed with Lambda Stack. It's loved by thousands of **Lambda customers**.

Lambda Stack is both a system wide package, a Dockerfile, and a Docker image.

Lambda Stack is not only a system wide installation of all of your favorite frameworks and drivers but also **a convenient "everything included" deep learning Docker image**. Now you'll have your team up and running with **GPU-accelerated Docker images in minutes** instead of weeks. To learn more about **how to set up Lambda Stack GPU Dockerfiles** check out our tutorial:

https://lambdalabs.com/blog/set-up-a-tensorflow-gpu-docker-container-using-lambda-stack-dockerfile/

```
# Build a Docker image for Ubuntu 20.04 (focal).
# You can substitute focal for bionic or xenial to change the ubuntu version.
sudo docker build \
    -t lambda-stack:20.04 \
    -f Dockerfile.focal \
    git://github.com/lambdal/lambda-stack-dockerfiles.git
```

- Lambda Stack Dockerfiles

Building images

Build the image with the appropriate command for the distribution you wish to use.

```
sudo docker build -t lambda-stack:16.04 -f Dockerfile.xenial .
sudo docker build -t lambda-stack:18.04 -f Dockerfile.bionic .
sudo docker build -t lambda-stack:20.04 -f Dockerfile.focal .
```

lambda-stack-dockerfiles/Dockerfile.focal

```
FROM ubuntu:20.04

WORKDIR /root/

# Add libcuda dummy dependency
ADD control .
RUN apt-get update && \
  DEBIAN_FRONTEND=noninteractive apt-get install --yes equivs && \
  equivs-build control && \
  dpkg -i libcuda1-dummy_11.1_all.deb && \
  rm control libcuda1-dummy_11.1_all.deb && \
  apt-get remove --yes --purge --autoremove equivs && \
  rm -rf /var/lib/apt/lists/*

# Setup Lambda repository
ADD lambda.gpg .
RUN apt-get update && \
  apt-get install --yes gnupg && \
  apt-key add lambda.gpg && \
  rm lambda.gpg && \
  echo "deb http://archive.lambdalabs.com/ubuntu focal main" > /etc/apt/sources.list
    .d/lambda.list && \
  echo "Package: *" > /etc/apt/preferences.d/lambda && \
  echo "Pin: origin archive.lambdalabs.com" >> /etc/apt/preferences.d/lambda && \
  echo "Pin-Priority: 1001" >> /etc/apt/preferences.d/lambda && \
  echo "cudnn cudnn/license_preseed select ACCEPT" | debconf-set-selections && \
  apt-get update && \
  DEBIAN_FRONTEND=noninteractive \
    apt-get install \
    --yes \
    --no-install-recommends \
    --option "Acquire::http::No-Cache=true" \
    --option "Acquire::http::Pipeline-Depth=0" \
    lambda-stack-cuda \
    lambda-server && \
  rm -rf /var/lib/apt/lists/*

# Setup for nvidia-docker
ENV NVIDIA_VISIBLE_DEVICES all
ENV NVIDIA_DRIVER_CAPABILITIES compute,utility
ENV NVIDIA_REQUIRE_CUDA "cuda>=10.2"
```

4.5 Automation for using Docker and Docker Compose

4.5.1 Makefile for using Docker and Docker Compose

- `4-docker/docker/Makefile`

```
# ##############################################################################
# Makefile for docker and docker-compose
# ##############################################################################

# Install GNU Make for Windows
# https://chocolatey.org/packages/make
# choco install -y make

# ##############################################################################

# ==============================================================================
# Include and Set Variables
# ==============================================================================

# include environment.env file if exists
-include ../environment.env

ifndef ENV
    ENV = dev
endif

DOCKER_COMPOSE = docker-compose \
  -f docker-compose.${ENV}.yml

CI_REGISTRY_IMAGE = registry.gitlab.com/siaiedu/python-on-linux

# ==============================================================================
# make all and restart
# ==============================================================================

all:
  make clean ls up db

restart:
  make down up

# ==============================================================================
# List Containers and Volumes
# ==============================================================================

ls:
  docker image ls
  docker container ls -a
  docker volume ls
  docker network ls

# ==============================================================================
# Down and Cleanup
# ==============================================================================

prune:
  docker system prune

prune-image:
  docker image prune

prune-volume:
```

```
  docker volume prune

stop:
  FOR /f "tokens=*" %%i IN ('docker container ls -aq') DO docker container stop %%i

down:
  ${DOCKER_COMPOSE} down

clean:
  ${DOCKER_COMPOSE} down --remove-orphans --volumes
  docker container prune -f
  docker volume prune -f
  docker network prune -f

clean-container:
  FOR /f "tokens=*" %%i IN ('docker ps -aq') DO docker rm %%i

clean-image:
  FOR /f "tokens=*" %%i IN ('docker images --format "{{.ID}}"') DO docker rmi %%i

# ================================================================================
# Up and Running Docker Containers
# ================================================================================

# Run Docker Compose
up:
  ${DOCKER_COMPOSE} up -d --build

# create migrations directory:
#   docker exec -it app python /manage.py db init
# create migration script and alembic_version table:
#   docker exec -it app python /manage.py db migrate
# create user defined tables:
#   docker exec -it app python /manage.py db upgrade
db:
  docker exec -it app python /manage.py db init
  docker exec -it app python /manage.py db migrate
  docker exec -it app python /manage.py db upgrade

# ================================================================================
# Build and Push Docker Image
# ================================================================================

# Log in to GitLab's Container Registry
login:
  docker login registry.gitlab.com

# Build Docker Image
build:
  docker build -t ${CI_REGISTRY_IMAGE} -f ./Dockerfile .

# Push Docker Image
push:
  docker push ${CI_REGISTRY_IMAGE}

# ================================================================================
# Pull and Run Docker Image
# ================================================================================

# Pull Docker Image
pull:
  docker pull ${CI_REGISTRY_IMAGE}:latest

```

```
121 # Run Docker Image
122 run:
123   docker run --name python-on-linux -p 8888:8888 -d ${CI_REGISTRY_IMAGE}:latest
```

Appendix A

Appendices

A.1 Visual Studio Code (VS Code) Setups

A.1.1 Installing VS Code on Windows

Installing Prerequisites: chocolatey

`1-sys-deps.bat`

- Installing Windows Package Manager: Chocolatey
  INSTALLING CHOCOLATEY, Search Chocolatey Packages

```
23 @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
       InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.
       WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%
       PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

Installing VS Code on Windows using chocolatey

`1-python-on-windows/setups/3-vscode.bat`

- Installing VS Code using Chocolatey
  Visual Studio Code (Chocolatey Package)

```
17 choco install -y vscode
```

A.1.2 Configuring VS Code

Display Language

If your VS code display language is not English, please change the display language to English.

```
The Command Palette (Ctrl+Shift+P or F1)
> configure display language
> Select Display Language > en
```

Word Wrap

```
File > Preferences > Settings
> Search settings: "word wrap"
> Editor: Word Wrap
> Controls how lines should wrap. > on
```

Run Selected Text

```
File > Preferences > Keyboard Shortcuts
(or the Gear icon in the Activity Bar > Keyboard Shortcuts)
> Type to search in keybindings: "workbench.action.terminal.runSelectedText"
> double click "Terminal: Run Selected Text In Active Terminal"
> Press desired key combination and then press ENTER: Ctrl + r
```

Add below code in the `keybindings.json`:

```
[
    {
        "key": "ctrl+r",
        "command": "workbench.action.terminal.runSelectedText",
        "when": "editorTextFocus == true"
    }
]
```

A.1.3 Setting up VS Code Extensions

- Installing VS Code Extensions using Command line
  Command line extension management

```
code --install-extension (<extension-id> | <extension-vsix-path>)
```

VS Code Extensions for General Development

- TODO Highlight: **# TODO:** , **# FIXME:** , **# NOTICE:**

```
code --install-extension wayou.vscode-todo-highlight
```

Configuring VS Code for TODO Highlight

```
File > Preferences > Settings
> Search settings: "TODO Highlight"
> Todohighlight: Keywords > Edit in settings.json
```

`settings.json:`

```
{
    "todohighlight.keywords": [
        "NOTICE:",
    ],
}
```

VS Code Extensions for Python Development

- Pylance
- Python Extension Pack
- Python Extended (Snippets)
- Python Indent
- AREPL for python
- Python Docstring Generator
- indent-rainbow

VS Code Extensions for Documentation

- Markdown
  - Markdown Theme Kit
  - Markdown All in One
  - Markdown Shortcuts
  - markdownlint
  - Markdown Emoji
  - Markdown+Math
  - Markdown Preview Github Styling
  - Markdown Preview with Bitbucket Styles
  - Markdown Preview Include Files
- LaTeX Workshop
  - Unicode Latex
  - LaTeX Workshop
- PlantUML
  - PlantUML
  - Graphviz (dot) language support for Visual Studio Code

VS Code Extensions for Linux

- Bash
  - Bash IDE
  - Shebang Snippets
  - shell-format
- Remote Development
  - Remote - WSL
  - Remote - SSH

VS Code Extensions for Git

- Git Extension Pack
- Git Graph
- gitflow

VS Code Extensions for Docker

- Remote Development
  - Remote - Containers

A.2 Setting up LaTeX on Windows: Tex Live and LaTeX Workshop

A.2.1 Understanding TeX Distributions and Engines

Understanding TeX Distributions

- Getting LaTeX

> **Getting LaTeX**
>
> LaTeX is **free software** under the terms of the **LaTeX Project Public License (LPPL)**. LaTeX is distributed through **CTAN servers** or comes as part of many easily installable and usable **TeX distributions provided by the TeX User Group (TUG)** or third parties. If you run into trouble, visit the help section.
>
> **TeX Distributions**
>
> If you're new to TeX and LaTeX or just want an easy installation, **get a full TeX distribution**. The TeX Users Group (TUG) has a list of notable distributions that are entirely, or least primarily, free software.
>
> - Linux
> - (**MiKTeX**)
> - **TeX Live**
> - macOS
> - MacTeX
> - (**MiKTeX**)
> - Windows
> - **MiKTeX**
> - proTeXt
> - **TeX Live**
> - LaTeX online services
> - Overleaf: The easy to use, online, **collaborative** LaTeX editor
> - ShareLaTeX: ShareLaTeX is now part of Overleaf
> - Papeeria
> - LaTeX base
> - Datazar

Understanding TeX Engines

- TeX engines and extensions

> - e-TeX(extended TeX). Required by current LaTeX, incorporated in all common executables except tex itself (run etex for plain e-TeX). e-TeX manual.
> - **pdfTeX**, a TeX extension which **can directly produce PDF output** as well as DVI. Incorporates e-TeX.
> - **XeTeX** (FAQ), a TeX implementation support for **Unicode and system fonts**.

- LuaTeX

> **Welcome**
>
> **LuaTeX is an extended version of pdfTeX** using **Lua as an embedded scripting language**. The LuaTeX project's main objective is to provide an open and configurable variant of TeX while at the same time offering downward compatibility. From the user perspective we have
>
> - **pdfTeX** as stable and more or less frozen 8 bit engine,
> - **XeTeX** as **unicode input and font aware engine** using libraries for font handling,
> - and **LuaTeX** as **engine that is programmable and delegates as much as possible to Lua**, with the objective to keep the core engine lean and mean.
>
> Each engine has its benefits and drawbacks (more details here).

Understanding Bibliography management

- Bibliography management with bibtex

> LaTeX supports bibliographies out of the box, either embedding the references in your document or storing them in an external file. This article explains how to manage bibliography with the thebibliography environment and the **BibTeX** system.

A.2.2 Setting up Tex Live on Windows

(1) Download TeX Live

- Installing TeX Live over the Internet
- Acquiring TeX Live as an ISO image

(2) Execute `install-tl-windows.exe` as **Administrator**

(3) Configure installation settings as following:

```
TeX Live 2021 Installer
> Advanced
> Selections
  > Scheme > Change
    > Schemes > full scheme (everything)
  > N. of collections > Customize > Collections
    > Languages
      > Korean
      > US and UK English
    > Other collections > All
```

A.2.3 Setting up LaTeX Workshop on Windows

- Visual Studio Code LaTeX Workshop Extension

```
File > Preferences > Settings
  > Search settings:
    > "latex-workshop.latex.tools"
      > Latex-workshop > Latex: Tools
      > Edit in settings.json
    > "latex-workshop.latex.recipes"
      > Latex-workshop > Latex: Recipes
      > Edit in settings.json
```

`%USERPROFILE%\AppData\Roaming\Code\User\settings.json`

```
{
    "latex-workshop.latex.recipes": [
        {
            "name": "xelatex",
            "tools": [
                "xelatex"
            ]
        },
        {
            "name": "lualatex",
            "tools": [
                "lualatex"
            ]
        },
        {
            "name": "lualatex -> bibtex -> lualatex*2",
            "tools": [
                "lualatex",
                "bibtex",
                "lualatex",
                "lualatex"
            ]
        }
    ],
    "latex-workshop.latex.tools": [
        {
            "name": "latexmk",
            "command": "latexmk",
            "args": [
                "-xelatex",
                "-shell-escape",
                "-synctex=1",
                "-interaction=nonstopmode",
                "-file-line-error",
                "-pdf",
                "-outdir=%OUTDIR%",
                "%DOC%"
            ],
            "env": {}
        },
        {
            "name": "xelatex",
            "command": "xelatex",
            "args": [
                "-shell-escape",
                "-synctex=1",
                "-interaction=nonstopmode",
                "-file-line-error",
                "-outdir=%OUTDIR%",
                "%DOC%"
```

```
            ],
            "env": {}
        },
        {
            "name": "lualatexmk",
            "command": "latexmk",
            "args": [
                "-synctex=1",
                "-interaction=nonstopmode",
                "-file-line-error",
                "-lualatex",
                "-outdir=%OUTDIR%",
                "%DOC%"
            ],
            "env": {}
        },
        {
            "name": "pdflatex",
            "command": "pdflatex",
            "args": [
                "-shell-escape",
                "-synctex=1",
                "-interaction=nonstopmode",
                "-file-line-error",
                "-outdir=%OUTDIR%",
                "%DOC%"
            ],
            "env": {}
        },
        {
            "name": "bibtex",
            "command": "bibtex",
            "args": [
                "%DOCFILE%"
            ],
            "env": {}
        }
    ]
}
```

References

[1] S. Chacon. *Pro Git*. Expert's voice in software development. Apress, 2009. ISBN: 9781430218340. URL: https://git-scm.com/book/en/v2.

[2] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* (2021). (Accelerated article preview). DOI: 10.1038/s41586-021-03819-2.

[3] Microsoft. *Python extension for Visual Studio Code*. URL: https://marketplace.visualstudio.com/items?itemName=ms-python.python.