In [ ]:

```
#Import the necessary liabilities
import os
import re
from sklearn.model_selection import train_test_split
from collections import Counter
import numpy as np
from sklearn.linear_model import RidgeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt
```

In [34]:

```python
'''
Text loading and preprocessing part.
Transforming the file into text which only contains lower alphabet and space.
'''

# Load and preprocess the text data
def load_and_preprocess_data(data_folder):
    texts = []
    labels = []

    for language_folder in os.listdir(data_folder):
        language_path = os.path.join(data_folder, language_folder)
        if os.path.isdir(language_path):
            for file_name in os.listdir(language_path):
                if "sentences" in file_name:
                    count=0
                    file_path = os.path.join(language_path, file_name)
                    with open(file_path, 'r', encoding='utf-8') as file:
                        for f in file:
                            if count < 100:
                                text=f.strip().split('\t', 1)
                                text=re.sub(r'[^A-Za-z]', '', str(text))
                                text=text.lower()
    #                           text = file.read().lower()  # Convert to lowercase
    #                           text = re.sub(r'[^a-z\s]', '', text)  # Remove non-alph
                                texts.append(text)
                                labels.append(file_name.split('_')[0])  # Use the fo
                                count+=1

    return texts, labels

# The path to the stores corpus data
data_folder = 'D://download//Language'

# Load and preprocess the data
texts, labels = load_and_preprocess_data(data_folder)

# Split the data into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, ra

# Print out the number of samples for verification
print(f"Number of training samples: {len(X_train)}")
print(f"Number of testing samples: {len(X_test)}")
```

```
Number of training samples: 1400
Number of testing samples: 600
```

In [35]:
```
'''
This part is set to generate the trigrams from given preprocessed text.
'''
# Function to generate trigrams
def generate_trigrams(text):
    trigrams = [text[i:i+3] for i in range(len(text) - 2)]
    return trigrams
```

In [36]:
```
'''
This part is to transform generated trigrams into HD vector.
'''
# Encode trigrams into a high-dimensional vector
def encode_trigrams(trigrams, d):
    # First, initializing a vector of zeros
    hd_vector = np.zeros(d)

    # For each trigram, generate a random vector in {+1, -1} space
    for trigram in trigrams:
        np.random.seed(hash(trigram) % (2**32 - 1))  # Seed based on trigram for ran
        random_vector = np.random.choice([-1, 1], size=d)
        hd_vector += random_vector

    # Normalize the vector
    hd_vector = np.sign(hd_vector)

    return hd_vector
```

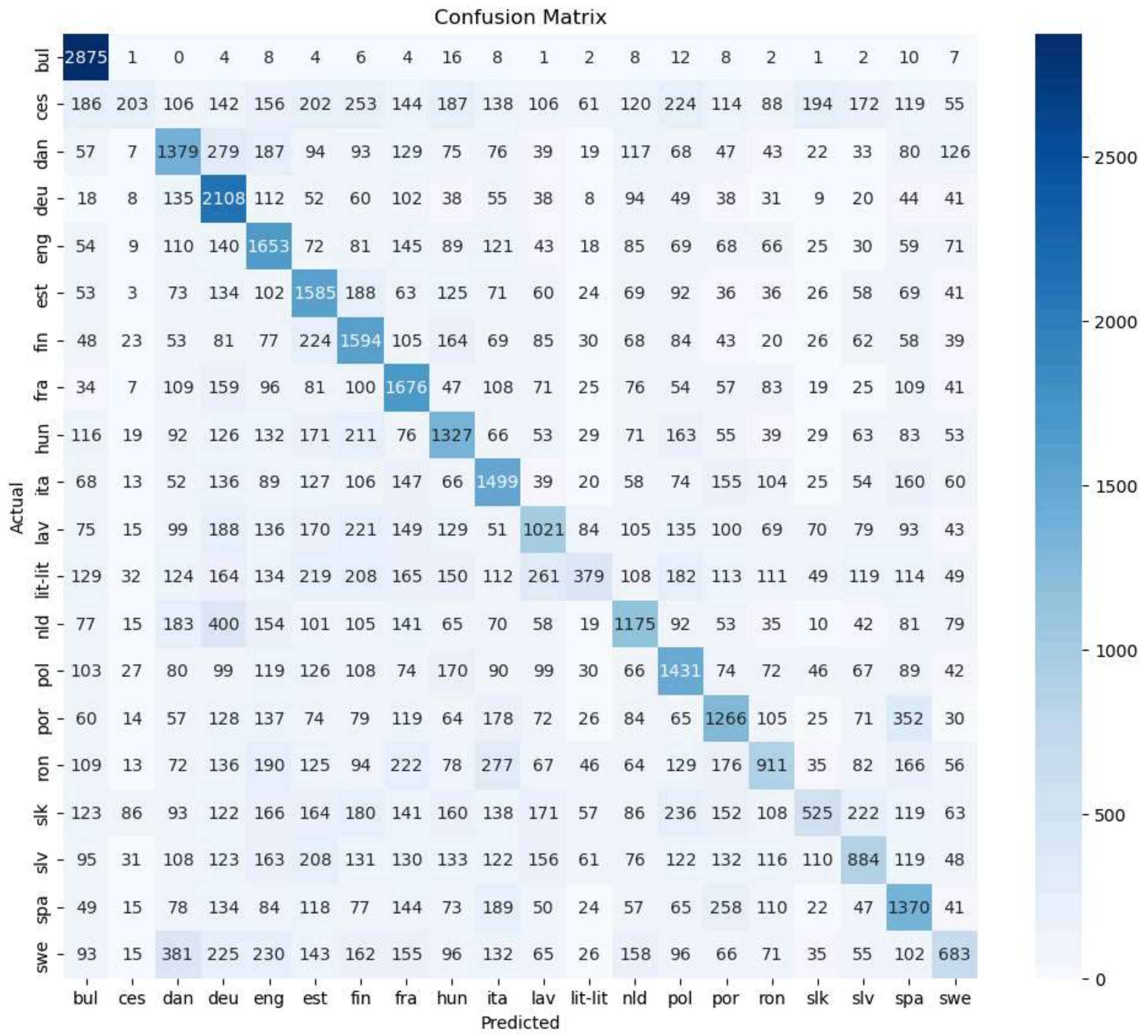
In [ ]:

```
'''
Training and Evaluating part on dimensionality of 100
This module is based on the data volume of 20*10,000
'''
# Encode the entire dataset to HD vectors
def encode_dataset(texts, d):
    encoded_vectors = []
    for text in texts:
        trigrams = generate_trigrams(text)
        encoded_vector = encode_trigrams(trigrams, d)
        encoded_vectors.append(encoded_vector)
    return np.array(encoded_vectors)

# Set dimensionality
d = 100

# Encode training and testing datasets
X_train_encoded = encode_dataset(X_train, d)
X_test_encoded = encode_dataset(X_test, d)

# Print shapes to verify
print(f"Encoded training set shape: {X_train_encoded.shape}")
print(f"Encoded testing set shape: {X_test_encoded.shape}")
```

In [25]:

```
# Encode the language labels into integers
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Train the Ridge regression model
ridge_model = RidgeClassifier()
ridge_model.fit(X_train_encoded, y_train_encoded)

# Predict on the test set
y_pred = ridge_model.predict(X_test_encoded)

# Decode the predicted labels back to the original language labels
y_pred_labels = label_encoder.inverse_transform(y_pred)
```

```
In  [26]: # Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_labels, labels=label_encoder.classes_)
plt.figure(figsize=(12, 10))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Accuracy and F1-Score
accuracy = accuracy_score(y_test, y_pred_labels)
f1 = f1_score(y_test, y_pred_labels, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
```




```
Accuracy: 0.4257
F1-Score: 0.4026
```

In [4]:

```
'''
Training and Evaluating part on dimensionality of 1000
Due to time machine and memory restriction, this module is based on the data volume o
'''
# Encode the entire dataset to HD vectors
def encode_dataset(texts, d):
    encoded_vectors = []
    for text in texts:
        trigrams = generate_trigrams(text)
        encoded_vector = encode_trigrams(trigrams, d)
        encoded_vectors.append(encoded_vector)
    return np.array(encoded_vectors)

# Set dimensionality
d = 1000

# Encode training and testing datasets
X_train_encoded = encode_dataset(X_train, d)
X_test_encoded = encode_dataset(X_test, d)

# Print shapes to verify
print(f"Encoded training set shape: {X_train_encoded.shape}")
print(f"Encoded testing set shape: {X_test_encoded.shape}")
```

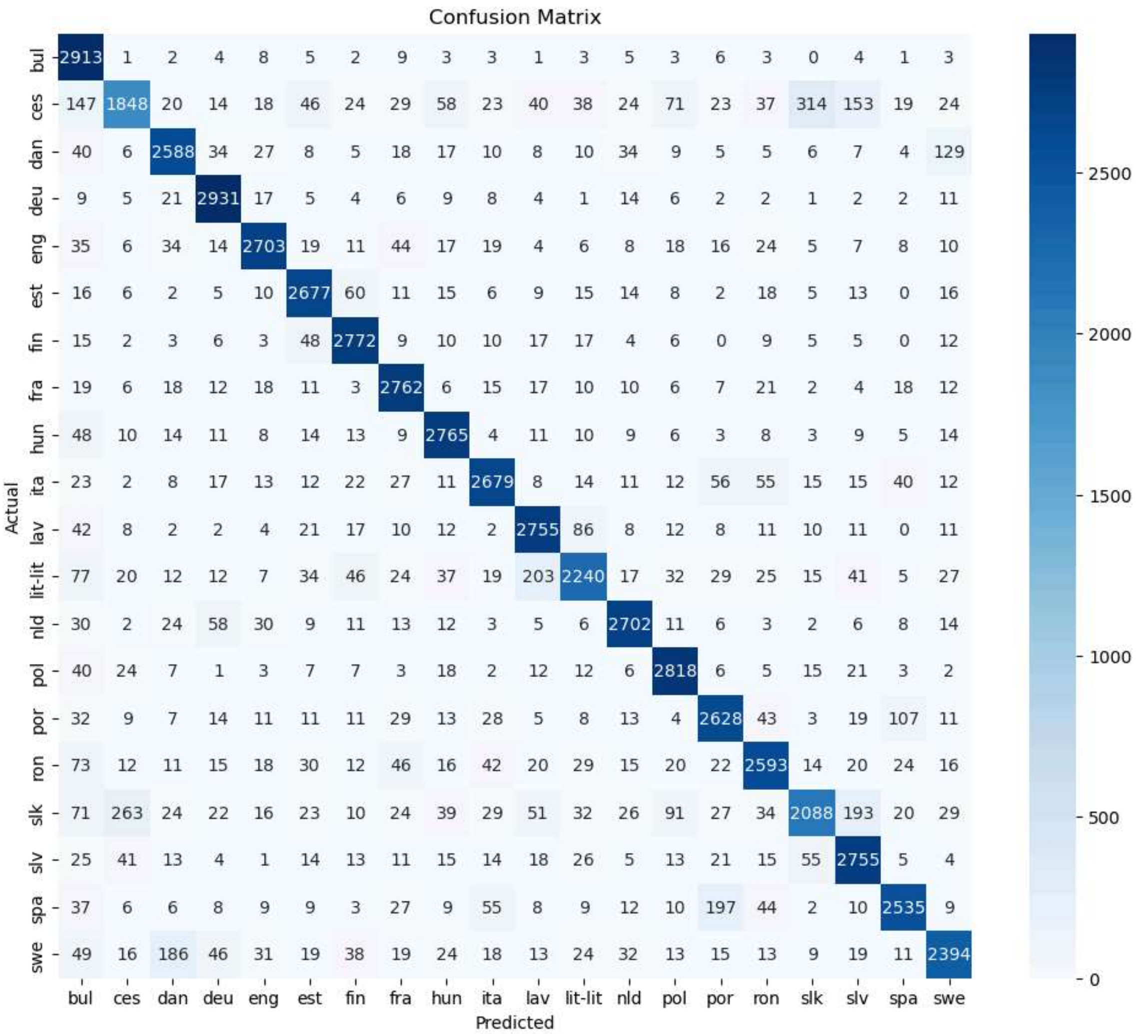
```
Encoded training set shape: (140000, 1000)
Encoded testing set shape: (60000, 1000)
```

In [5]:

```
# Encode the language labels into integers
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Train the Ridge regression model
ridge_model = RidgeClassifier()
ridge_model.fit(X_train_encoded, y_train_encoded)

# Predict on the test set
y_pred = ridge_model.predict(X_test_encoded)

# Decode the predicted labels back to the original language labels
y_pred_labels = label_encoder.inverse_transform(y_pred)
```

In [6]:

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_labels, labels=label_encoder.classes_)
plt.figure(figsize=(12, 10))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Accuracy and F1-Score
accuracy = accuracy_score(y_test, y_pred_labels)
f1 = f1_score(y_test, y_pred_labels, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
```



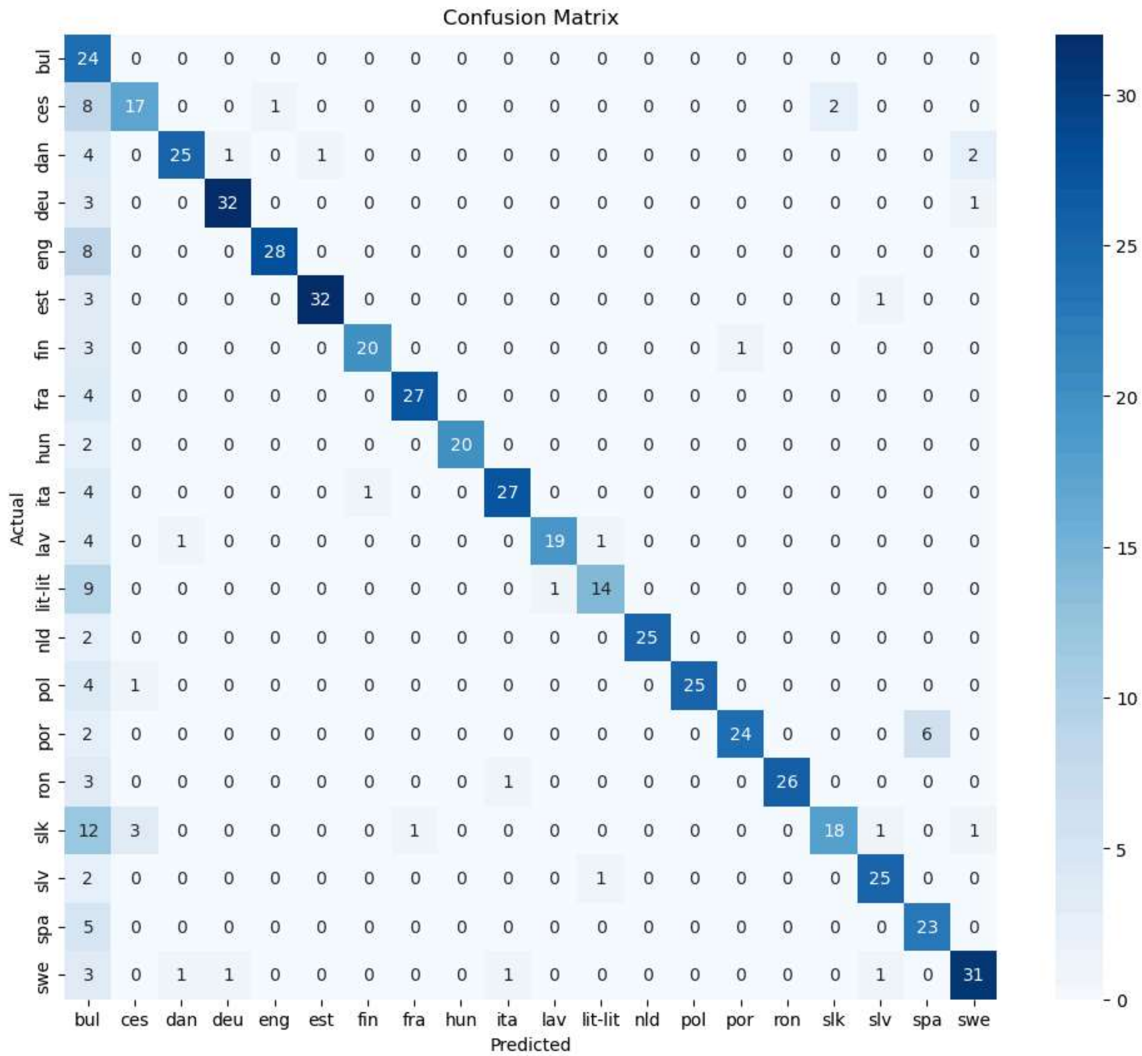

```
Accuracy: 0.8691
F1-Score: 0.8670
```

In [37]:
```python
'''
Training and Evaluating part on dimensionality of 10000
Due to time machine and memory restriction, this module is based on the data volume o
'''

#Encode the entire dataset
def encode_dataset(texts, d):
    encoded_vectors = []
    for text in texts:
        trigrams = generate_trigrams(text)
        encoded_vector = encode_trigrams(trigrams, d)
        encoded_vectors.append(encoded_vector)
    return np.array(encoded_vectors)

# Set dimensionality
d = 10000

# Encode training and testing datasets
X_train_encoded = encode_dataset(X_train, d)
X_test_encoded = encode_dataset(X_test, d)

# Print shapes of training and testing data shape
print(f"Encoded training set shape: {X_train_encoded.shape}")
print(f"Encoded testing set shape: {X_test_encoded.shape}")
```

```
Encoded training set shape: (1400, 10000)
Encoded testing set shape: (600, 10000)
```

In [38]:
```python
# Encode the language labels into integers
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Train the Ridge regression model
ridge_model = RidgeClassifier()
ridge_model.fit(X_train_encoded, y_train_encoded)

# Predict on the test set
y_pred = ridge_model.predict(X_test_encoded)

# Decode the predicted labels back to the original language labels
y_pred_labels = label_encoder.inverse_transform(y_pred)
```

In [39]:
```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_labels, labels=label_encoder.classes_)
plt.figure(figsize=(12, 10))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Accuracy and F1-Score
accuracy = accuracy_score(y_test, y_pred_labels)
f1 = f1_score(y_test, y_pred_labels, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
```




```
Accuracy: 0.8033
F1-Score: 0.8348
```

In [ ]: