The command !git clone https://github.com/google-research/bert.git is used to copy the contents of the BERT (Bidirectional Encoder Representations from Transformers) repository from GitHub to your local environment. The exclamation mark allows this command to be run in environments like Jupyter Notebook or Google Colab, while git clone retrieves all files, history, and branches from the specified repository. This command is essential for downloading and working with the official BERT implementation from Google's research team.

```
!git clone https://github.com/google-research/bert.git explain this code
```

```
Cloning into 'bert'...
remote: Enumerating objects: 340, done.
remote: Counting objects: 100% (340/340), done.
remote: Compressing objects: 100% (154/154), done.
remote: Total 340 (delta 203), reused 303 (delta 185), pack-reused 0
Receiving objects: 100% (340/340), 192.58 KiB | 5.35 MiB/s, done.
Resolving deltas: 100% (203/203), done.
```

```
%cd bert
```

```
/content/bert
```

This code snippet loads the MRPC (Microsoft Research Paraphrase Corpus) dataset from the datasets library and saves the training and validation splits as TSV (Tab-Separated Values) files. First, it checks if a data directory exists and creates it if necessary. The dataset is then converted into pandas DataFrames for easier manipulation. The relevant columns (sentence1, sentence2, and label) are extracted and saved as train.tsv and dev.tsv files within the data directory, ensuring the files are formatted without headers. Finally, a confirmation message is printed to indicate that the data has been saved successfully.

```
from datasets import load_dataset
import os

# Load the MRPC dataset
dataset = load_dataset("glue", "mrpc")

# Create the 'data' directory if it doesn't exist
os.makedirs('data', exist_ok=True)

# Convert datasets to pandas DataFrames
train_df = dataset['train'].to_pandas()
val_df = dataset['validation'].to_pandas()

# Save the data as TSV files
train_df[['sentence1', 'sentence2', 'label']].to_csv('data/train.tsv', index=False, header=False, sep='\t')
val_df[['sentence1', 'sentence2', 'label']].to_csv('data/dev.tsv', index=False, header=False, sep='\t')

print("Data saved successfully.")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Downloading readme: 100%    35.3k/35.3k [00:00<00:00, 619kB/s]
Downloading data: 100%    649k/649k [00:00<00:00, 1.93MB/s]
Downloading data: 100%    75.7k/75.7k [00:00<00:00, 323kB/s]
Downloading data: 100%    308k/308k [00:00<00:00, 564kB/s]
Generating train split: 100%    3668/3668 [00:00<00:00, 59740.64 examples/s]
Generating validation split: 100%    408/408 [00:00<00:00, 9856.79 examples/s]
Generating test split: 100%    1725/1725 [00:00<00:00, 33544.94 examples/s]
Data saved successfully.
```

This code loads a pre-trained BERT model and its tokenizer for a binary sequence classification task. It uses the bert-base-uncased variant of BERT, which is case-insensitive. The BertTokenizer is loaded to convert text into token IDs, while the BertForSequenceClassification model is configured for binary classification with two output labels. This setup allows for fine-tuning the BERT model on specific tasks like determining if two sentences are paraphrases.

```
from transformers import BertTokenizer, BertForSequenceClassification

# Load pre-trained BERT model and tokenizer
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
tokenizer_config.json: 100%    48.0/48.0 [00:00<00:00, 2.37kB/s]
vocab.txt: 100%    232k/232k [00:00<00:00, 7.41MB/s]
tokenizer.json: 100%    466k/466k [00:00<00:00, 2.32MB/s]
config.json: 100%    570/570 [00:00<00:00, 41.2kB/s]
model.safetensors: 100%    440M/440M [00:01<00:00, 230MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initiali
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

This code loads the MRPC dataset using the datasets library and tokenizes it for use with a pre-trained BERT model. The dataset is tokenized by applying a custom tokenize_function, which uses the BERT tokenizer to process pairs of sentences (sentence1 and sentence2), truncating them and padding to a maximum length. The map function is then applied to the entire dataset, ensuring that the tokenization is done in batches, making the data ready for model training.

```
from transformers import TrainingArguments, Trainer
from transformers import BertTokenizer, BertForSequenceClassification
from datasets import load_dataset

# Load the dataset
dataset = load_dataset("glue", "mrpc")

# Tokenize the dataset
def tokenize_function(examples):
    return tokenizer(examples['sentence1'], examples['sentence2'], truncation=True, padding='max_length')

tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

```
Map: 100%    3668/3668 [00:04<00:00, 890.38 examples/s]
Map: 100%    408/408 [00:00<00:00, 827.36 examples/s]
Map: 100%    1725/1725 [00:02<00:00, 794.37 examples/s]
```

This code sets up the training configuration for fine-tuning a BERT model using the Trainer API from the transformers library. The TrainingArguments defines various training parameters, such as saving outputs to the ./results directory, running the training for 3 epochs, and using a batch size of 8 for both training and evaluation. It also includes 500 warmup steps to gradually adjust the learning rate, applies a weight decay of 0.01 to prevent overfitting, and logs training progress every 10 steps in the ./logs directory. The Trainer class is then instantiated with the pre-trained BERT model, the defined training arguments, and the tokenized training and evaluation datasets. This setup streamlines the process of fine-tuning the BERT model on the MRPC dataset.

```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=3,              # number of training epochs
    per_device_train_batch_size=8,   # batch size for training
    per_device_eval_batch_size=8,    # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    logging_steps=10,
)

trainer = Trainer(
    model=model,                         # the instantiated Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=tokenized_datasets['train'],         # training dataset
    eval_dataset=tokenized_datasets['validation']      # evaluation dataset
)


trainer.train()
```

[1377/1377 16:56, Epoch 3/3]

Step	Training Loss
10	0.780200
20	0.787400
30	0.762200
40	0.746700
50	0.707300
60	0.666500
70	0.637600
80	0.629300
90	0.581200
100	0.683100
110	0.617700
120	0.574400
130	0.511200
140	0.605500
150	0.600200
160	0.616400
170	0.578700
180	0.557600
190	0.539200
200	0.523200
210	0.583000
220	0.501100
230	0.502300
240	0.623200
250	0.439900
260	0.546800
270	0.550400
280	0.568400
290	0.594800
300	0.508500
310	0.526700
320	0.553500
330	0.421100
340	0.706300
350	0.530900
360	0.533800
370	0.576500
380	0.500600
390	0.499100
400	0.425300
410	0.377200
420	0.421900
430	0.555200
440	0.667000
450	0.595900

460	0.470700
470	0.445300
480	0.441100
490	0.372000
500	0.316300
510	0.379500
520	0.676000
530	0.533400
540	0.477900
550	0.377600
560	0.423500
570	0.360700
580	0.361600
590	0.248200
600	0.408000
610	0.302600
620	0.391300
630	0.225700
640	0.635200
650	0.370800
660	0.389100
670	0.596900
680	0.530800
690	0.341700
700	0.305200
710	0.448200
720	0.472500
730	0.375900
740	0.341500
750	0.555900
760	0.400100
770	0.239400
780	0.361900
790	0.334900
800	0.435800
810	0.495400
820	0.339400
830	0.280800
840	0.433500
850	0.470400
860	0.454800
870	0.340000
880	0.383800
890	0.434400
900	0.446800
910	0.430700
920	0.184100

930	0.178200
940	0.077400
950	0.213600
960	0.208100
970	0.358100
980	0.062000
990	0.222800
1000	0.127400
1010	0.120100
1020	0.146100
1030	0.194200
1040	0.364500
1050	0.234300
1060	0.396100
1070	0.283800
1080	0.226900
1090	0.112300
1100	0.086000
1110	0.202400
1120	0.100700
1130	0.219000
1140	0.142600
1150	0.047300
1160	0.265600
1170	0.151800
1180	0.022000
1190	0.244700
1200	0.327100
1210	0.330700
1220	0.308500
1230	0.276000
1240	0.207500
1250	0.130000
1260	0.177400
1270	0.286800
1280	0.327200
1290	0.242800
1300	0.117500
1310	0.203900
1320	0.221600
1330	0.067500
1340	0.164100
1350	0.244800
1360	0.324800
1370	0.122400

```
TrainOutput(global_step=1377, training_loss=0.39418783539347485, metrics={'train_runtime': 1018.4023, 'train_samples_per_second':
10.805, 'train_steps_per_second': 1.352, 'total_flos': 2895274053181440.0, 'train_loss': 0.39418783539347485, 'epoch': 3.0})
```

```
eval_results = trainer.evaluate()
print(eval_results)
```

[51/51 00:10]

```
{'eval_loss': 0.6390026211738586, 'eval_runtime': 11.1132, 'eval_samples_per_second': 36.713, 'eval_steps_per_second': 4.589, 'epoch': 3
```

This code sets up a pipeline for fine-tuning a BERT model on the MRPC dataset. It begins by loading the MRPC dataset using the datasets library. The BertTokenizer is then initialized with the bert-base-uncased model, and a custom tokenize_function is defined to tokenize sentence pairs from the dataset with truncation and padding applied to ensure consistent input length. This tokenization is applied to the entire dataset in batches. Afterward, the BERT model is loaded, specifically configured for sequence classification with two output labels (since MRPC is a binary classification task). Finally, the code ensures that a directory named results exists to store the output of the training process, creating it if necessary. This setup is a complete preparation for fine-tuning the BERT model on the tokenized MRPC dataset.

```
from datasets import load_dataset
from transformers import BertTokenizer, BertForSequenceClassification, TrainingArguments, Trainer
import os

# Load the MRPC dataset
dataset = load_dataset("glue", "mrpc")

# Tokenize the dataset
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(examples['sentence1'], examples['sentence2'], truncation=True, padding='max_length')

tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Load the BERT model
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Create the 'results' directory if it doesn't exist
os.makedirs('results', exist_ok=True)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

Downloading readme: 100%	35.3k/35.3k [00:00<00:00, 578kB/s]
Downloading data: 100%	649k/649k [00:00<00:00, 2.27MB/s]
Downloading data: 100%	75.7k/75.7k [00:00<00:00, 305kB/s]
Downloading data: 100%	308k/308k [00:00<00:00, 1.03MB/s]
Generating train split: 100%	3668/3668 [00:00<00:00, 48768.04 examples/s]
Generating validation split: 100%	408/408 [00:00<00:00, 6067.37 examples/s]
Generating test split: 100%	1725/1725 [00:00<00:00, 28068.66 examples/s]
tokenizer_config.json: 100%	48.0/48.0 [00:00<00:00, 1.64kB/s]
vocab.txt: 100%	232k/232k [00:00<00:00, 1.69MB/s]
tokenizer.json: 100%	466k/466k [00:00<00:00, 3.50MB/s]
config.json: 100%	570/570 [00:00<00:00, 8.13kB/s]
Map: 100%	3668/3668 [00:10<00:00, 319.90 examples/s]
Map: 100%	408/408 [00:01<00:00, 371.06 examples/s]
Map: 100%	1725/1725 [00:02<00:00, 689.22 examples/s]
model.safetensors: 100%	440M/440M [00:02<00:00, 205MB/s]

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initiali
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

This code configures and runs a training process for a BERT model with an increased learning rate of 5e-5. It sets up training arguments with a batch size of 16 and saves results and logs in specified directories. The Trainer class is used to train the model and evaluate its performance on the MRPC dataset with these new settings, and the evaluation results are printed.

```
training_args_lr = TrainingArguments(
    output_dir='./results_lr',          # output directory
    num_train_epochs=3,                 # number of training epochs
    per_device_train_batch_size=16,     # batch size for training
    per_device_eval_batch_size=16,      # batch size for evaluation
    learning_rate=5e-5,                 # increased learning rate
    warmup_steps=500,                   # number of warmup steps for learning rate scheduler
    weight_decay=0.01,                  # strength of weight decay
    logging_dir='./logs_lr',            # directory for storing logs
    logging_steps=10,
)

trainer_lr = Trainer(
    model=model,                         # the instantiated Transformers model to be trained
    args=training_args_lr,               # training arguments, defined above
    train_dataset=tokenized_datasets['train'],         # training dataset
    eval_dataset=tokenized_datasets['validation']      # evaluation dataset
)

# Train with the increased learning rate
trainer_lr.train()
eval_results_lr = trainer_lr.evaluate()
print("Learning Rate Adjustment Results:", eval_results_lr)
```

Step	Training Loss
10	0.736900
20	0.746700
30	0.713200
40	0.695200
50	0.696100
60	0.669500
70	0.626900
80	0.646800
90	0.610000
100	0.594100
110	0.583500
120	0.562000
130	0.534300
140	0.577300
150	0.591000
160	0.624900
170	0.536800
180	0.552100
190	0.539700
200	0.545600
210	0.487100
220	0.532100
230	0.574400
240	0.423000
250	0.407800
260	0.427000
270	0.486700
280	0.485100
290	0.399300
300	0.384200
310	0.325900
320	0.447900
330	0.554800
340	0.438600
350	0.418500
360	0.393600
370	0.337500
380	0.445100
390	0.427700
400	0.280500
410	0.432300
420	0.386800
430	0.467600
440	0.352200
450	0.527700