In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as io
from scipy.io import loadmat
import math
from sklearn.model_selection import train_test_split
import random
%matplotlib inline
```

In [16]:
```
np.random.seed(671)
random.seed(671)

minist_data = loadmat('mnist-original.mat')

# train, test = train_test_split(minist_data, test_size = 1/7, random_state =
 671)

# X_train_pool = train['data'].transpose()
# y_train_pool = train['label'].reshape(-1)
# X_test_pool = test['data'].transpose()
# y_test_pool = test['label'].reshape(-1)

X = minist_data['data'].transpose()
y = minist_data['label'].reshape(-1)


rand_index = random.sample(range(0,len(X)),len(X))

i = 0
X_rand_lst = []
y_rand_lst = []
for index in rand_index:
    X_rand_lst.append(X[index])
    y_rand_lst.append(y[index])
    i = i+1

X_rand = np.asarray(X_rand_lst)

y_rand = np.asarray(y_rand_lst)



X_train_pool = X_rand[:60000,:]
y_train_pool = y_rand[:60000]
X_test_pool = X_rand[60000:,:]
y_test_pool = y_rand[60000:]

index = []
for label in set(y_train_pool):
    if label == 4 or label == 9:
        index_temp = []
        index_temp = np.argwhere(y_train_pool == label).reshape(-1)
        index += list(index_temp)

X_train_int = np.take(X_train_pool, index, axis = 0)
y_train = np.take(y_train_pool, index, axis = 0)

for i in range(len(y_train)):
    if y_train[i] == 4:
        y_train[i] = -1
    elif y_train[i] == 9:
        y_train[i] = 1

index = []
for label in set(y_test_pool):
```

```
    if label == 4 or label == 9:
        index_temp = []
        index_temp = np.argwhere(y_test_pool == label).reshape(-1)
        index += list(index_temp)

X_test_int = np.take(X_test_pool, index, axis = 0)
y_test = np.take(y_test_pool, index, axis = 0)

for i in range(len(y_test)):
    if y_test[i] == 4:
        y_test[i] = -1
    elif y_test[i] == 9:
        y_test[i] = 1

# normalization of X
X_train_lst = []
for i in range(len(X_train_int)):
    X_train_lst.append([])
    for j in range(len(X_train_int[0])):
        X_train_lst[i].append(float(X_train_int[i][j])/255)

X_test_lst = []
for i in range(len(X_test_int)):
    X_test_lst.append([])
    for j in range(len(X_test_int[0])):
        X_test_lst[i].append(float(X_test_int[i][j])/255)

X_train = np.asarray(X_train_lst)

X_test = np.asarray(X_test_lst)
```

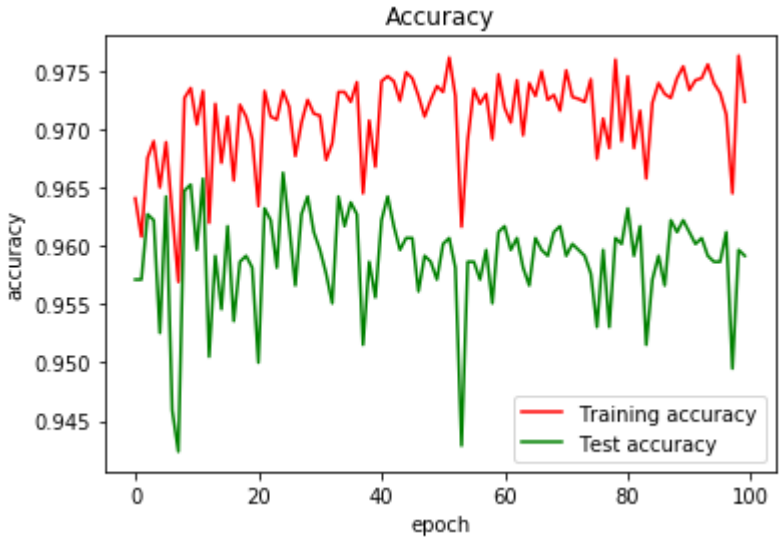
In [20]:

```
# shuffle the training set

rand_index = random.sample(range(0,len(X_train)),len(X_train))

i = 0
X_train_rand_lst = []
y_train_rand_lst = []
for index in rand_index:
    X_train_rand_lst.append(X_train[index])
    y_train_rand_lst.append(y_train[index])
    i = i+1

X_train_rand = np.asarray(X_train_rand_lst)

y_train_rand = np.asarray(y_train_rand_lst)
```

In [21]:
```
def perceptron(X, y, X_test, y_test, I = 100):

    # initilize w
    w = [0]*X.shape[1]
    acc_train = []
    acc_test = []
    for e in range(I):

        # perceptron
        for i in range(len(X)):
            if y[i] * np.matmul(w, X[i]) <= 0 or (i == 0 and e == 0):
                for j in range(len(w)):
                    w[j] = w[j] + y[i]*X[i][j]

        # Accuracy of training
        false_count = 0
        for i in range(len(X)):
            if y[i] * np.matmul(w, X[i]) <= 0:
                false_count = false_count + 1
        acc_temp = (len(X) - false_count) / len(X)
        acc_train.append(acc_temp)

        # Accuracy of test set
        false_count = 0
        for i in range(len(X_test)):
            if y_test[i] * np.matmul(w, X_test[i]) <= 0:
                false_count = false_count + 1
        acc_temp = (len(X_test) - false_count) / len(X_test)
        acc_test.append(acc_temp)
#         if e % 10 == 0:
#             print(e, end = '')
    return w, acc_train, acc_test
```

In [22]:
```
def plot_accuracy(acc_train, acc_test, I):
    x_axis = []
    for i in range(I):
        x_axis.append(i)
    plt.plot(x_axis,acc_train,'r',label='Training accuracy')
    plt.plot(x_axis,acc_test,'g',label='Test accuracy')
    plt.title('Accuracy')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.legend()
    plt.show()
```

In [23]:
```
w, acc_train, acc_test= perceptron(X_train_rand, y_train_rand, X_test, y_test,
100)
plot_accuracy(acc_train, acc_test, 100)
```



b and c

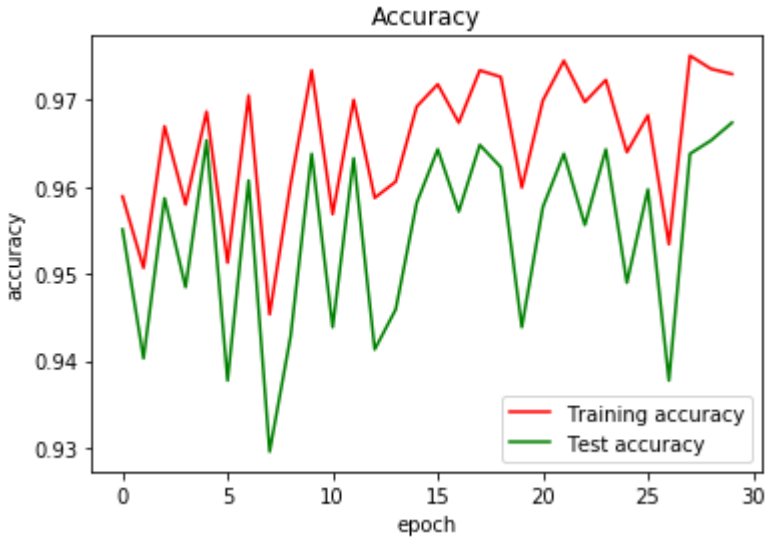
I observe that the training accuracy is relatively around 97%, and the testing accuracy is relatively around 96%. The testing accuracy is lower than the training accuracy.

When the maximum number of epochs increase, the training accuracy would slightly incearse, while the testing accuracy would slightly decrease. This shows a problem of overfitting.

In [24]:
```
def balanced_winnow(X, y, X_test, y_test, I = 30, eta = 0.1):

    # initilize w
    p = X.shape[1]
    w_p = [1/(2*p)]*X.shape[1]
    w_n = [1/(2*p)]*X.shape[1]
    acc_train = []
    acc_test = []
    w_p_all = []
    w_n_all = []
    for e in range(I):

    # balanced_winnow
        for i in range(len(X)):
            if y[i] * ( np.dot(w_p, X[i]) - np.dot(w_n, X[i]) ) <= 0 :
                for j in range(len(w_n)):
                    w_p[j] = w_p[j] * math.exp(eta*y[i]*X[i][j])
                    w_n[j] = w_n[j] * math.exp(-eta*y[i]*X[i][j])
                s = np.sum(w_p) + np.sum(w_n)
                for j in range(len(w_n)):
                    w_p[j] = w_p[j] / s
                    w_n[j] = w_n[j] / s
        w_p_all.append(w_p)
        w_n_all.append(w_n)

        # Accuracy of training
        false_count = 0
        for i in range(len(X)):
            if y[i] * ( np.dot(w_p, X[i]) - np.dot(w_n, X[i]) ) <= 0:
                false_count = false_count + 1
        acc_temp = (len(X) - false_count) / len(X)
        print(acc_temp, false_count)
        acc_train.append(acc_temp)

        # Accuracy of test set
        false_count = 0
        for i in range(len(X_test)):
            if y_test[i] * ( np.dot(w_p, X_test[i]) - np.dot(w_n, X_test[i]) )
<= 0:
                false_count = false_count + 1
        acc_temp = (len(X_test) - false_count) / len(X_test)
        acc_test.append(acc_temp)
        if e % 10 == 0:
            print(e, end = ' ')
    return w, acc_train, acc_test, w_p_all, w_n_all
```

In [25]:
```
w, acc_train, acc_test, w_p_all, w_n_all= balanced_winnow(X_train_rand, y_trai
n_rand, X_test, y_test)
plot_accuracy(acc_train, acc_test, 30)
```

```
0.9588902047030959 486
0 0.9506851632549485 583
0.9669260700389105 391
0.9579597360852647 497
0.9686178311622399 371
0.9512772796481137 576
0.9704787683979023 349
0.9453561157164608 646
0.9603282016579259 469
0.9733547623075621 315
0.9568600913551006 510
10 0.9699712400609034 355
0.958721028590763 488
0.9605819658264253 466
0.9692099475554051 364
0.9717475892403993 334
0.9673490103197429 386
0.9733547623075621 315
0.972593469802064 324
0.9599052613770935 474
0.9698866520047369 356
20 0.9744544070377262 302
0.969717475892404 358
0.972255117577398 328
0.9639654880730841 426
0.9681948908814075 376
0.9533919810522754 551
0.9750465234308916 295
0.9735239384198952 313
0.9729318220267298 320
```



e

When the data is shuffled properly, it only take one epch to reach a relatively stable state.

In [ ]: