In [55]:
```
# Define a function that takes an image, gradient orientation,
# and threshold min / max values.
def abs_sobel_thresh(img, orient='x', sobel_kernel=3, thresh=(0, 255)):
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Apply x or y gradient with the OpenCV Sobel() function
    # and take the absolute value
    if orient == 'x':
        abs_sobel = np.absolute(cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_ker
    if orient == 'y':
        abs_sobel = np.absolute(cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_ker
    # Rescale back to 8 bit integer
    scaled_sobel = np.uint8(255*abs_sobel/np.max(abs_sobel))
    # Create a copy and apply the threshold
    binary_output = np.zeros_like(scaled_sobel)
    # Here I'm using inclusive (>=, <=) thresholds, but exclusive is ok too
    binary_output[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])] = 1

    # Return the result
    return binary_output


# Define a function to return the magnitude of the gradient
# for a given sobel kernel size and threshold values
def mag_thresh(img, sobel_kernel=3, thresh=(0, 255)):
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Take both Sobel x and y gradients
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    # Calculate the gradient magnitude
    gradmag = np.sqrt(sobelx**2 + sobely**2)
    # Rescale to 8 bit
    scale_factor = np.max(gradmag)/255
    gradmag = (gradmag/scale_factor).astype(np.uint8)
    # Create a binary image of ones where threshold is met, zeros otherwise
    binary_output = np.zeros_like(gradmag)
    binary_output[(gradmag >= thresh[0]) & (gradmag <= thresh[1])] = 1

    # Return the binary image
    return binary_output

# Define a function to threshold an image for a given range and Sobel kernel
def dir_threshold(img, sobel_kernel=3, thresh=(0, np.pi/2)):
    # Grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Calculate the x and y gradients
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    # Take the absolute value of the gradient direction,
    # apply a threshold, and create a binary image result
    absgraddir = np.arctan2(np.absolute(sobely), np.absolute(sobelx))
    binary_output =  np.zeros_like(absgraddir)
    binary_output[(absgraddir >= thresh[0]) & (absgraddir <= thresh[1])] = 1

    # Return the binary image
```

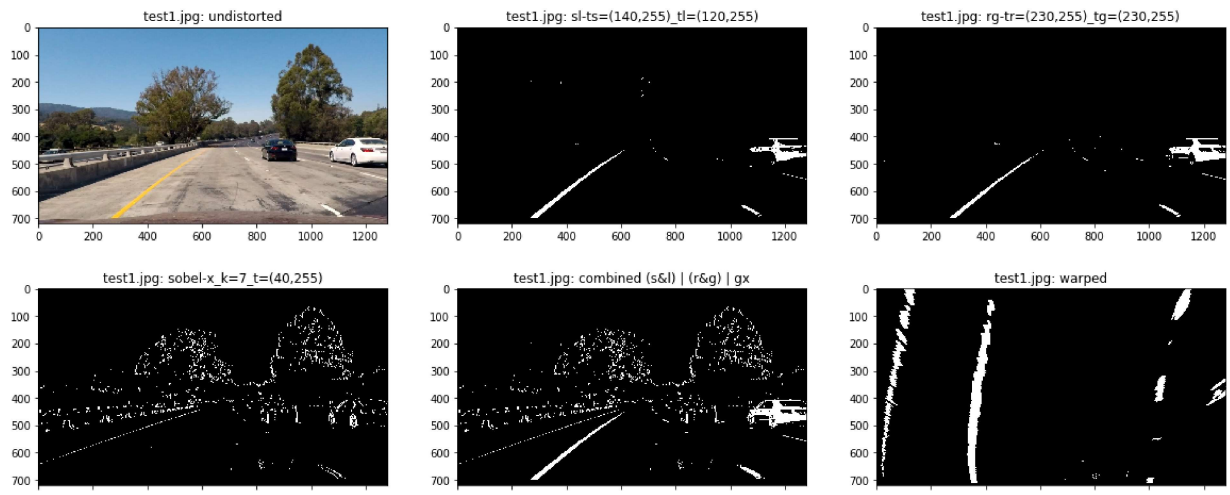
```
    return binary_output

# Define a function that thresholds the S-channel of HLS
def channel_select(img, channel=2, color_conversion = cv2.COLOR_BGR2HLS, thresh=(
    if color_conversion:
        c = cv2.cvtColor(img, color_conversion)
    else:
        c = img
    s_channel = c[:,:,channel]
    binary_output = np.zeros_like(s_channel)
    binary_output[(s_channel > thresh[0]) & (s_channel <= thresh[1])] = 1
    return binary_output

#combined = np.zeros_like(dir_binary)
#combined[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1))
#color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, s_binary)
```

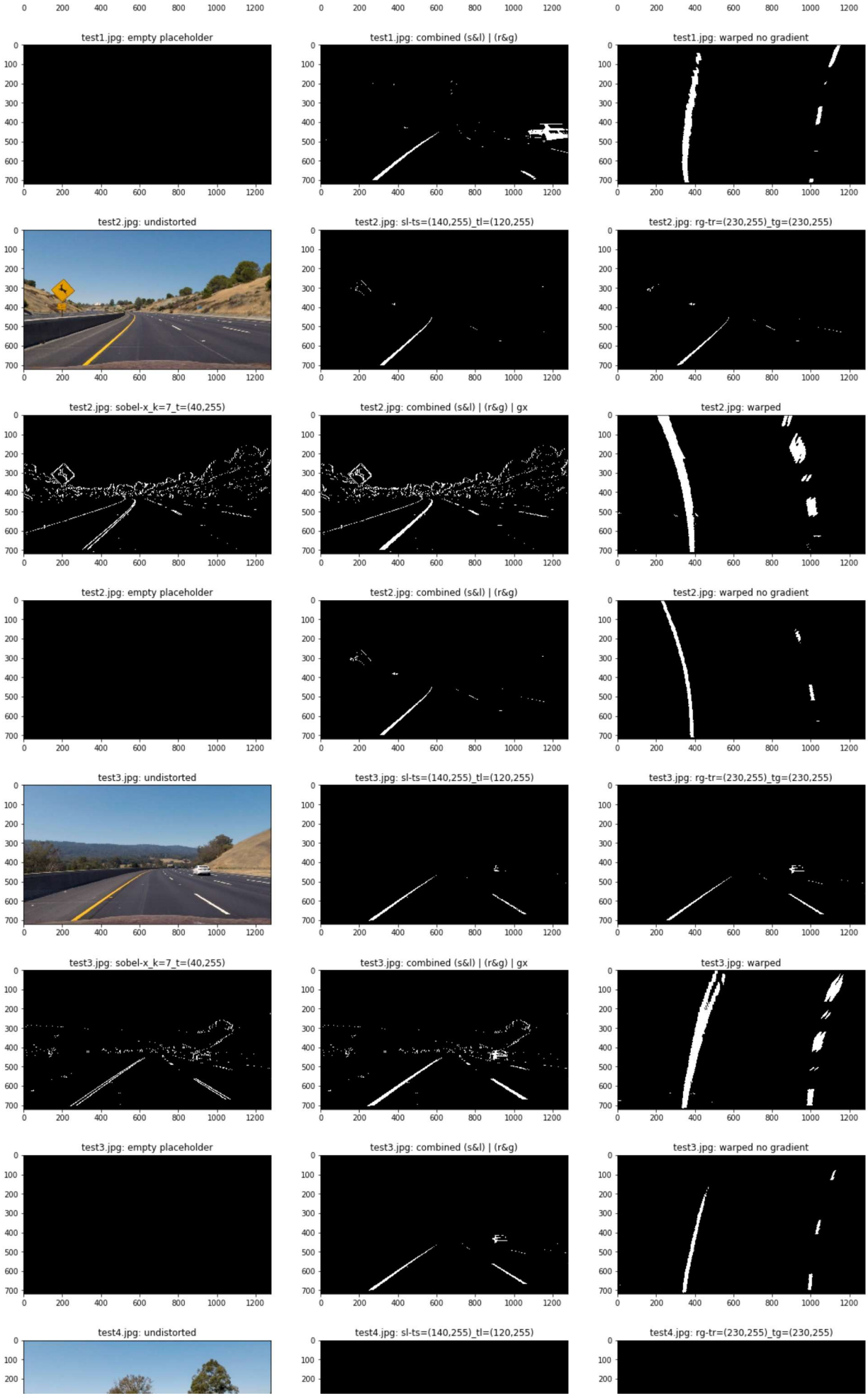
In [10]:
```
import os
import os.path

cam_calibration = pickle.load( open( "camera_calibration.p", "rb" ) )
print('Loaded camera calibration')
print('-------------------------')
print('mtx:\n', cam_calibration['mtx'])
print('\ndist:\n', cam_calibration['dist'])

def save_images(folder, images, names, binary=False):
    if not os.path.exists(folder):
        os.makedirs(folder)
    for img, fname in zip(images, names):
        p = os.path.join(folder, fname)
        if p[:-4] != '.jpg':
            p += '.jpg'
        if binary:
            img = img * 255
        cv2.imwrite(p, img)

def undistort_image(img):
    return cv2.undistort(img, cam_calibration['mtx'], cam_calibration['dist'], No
```

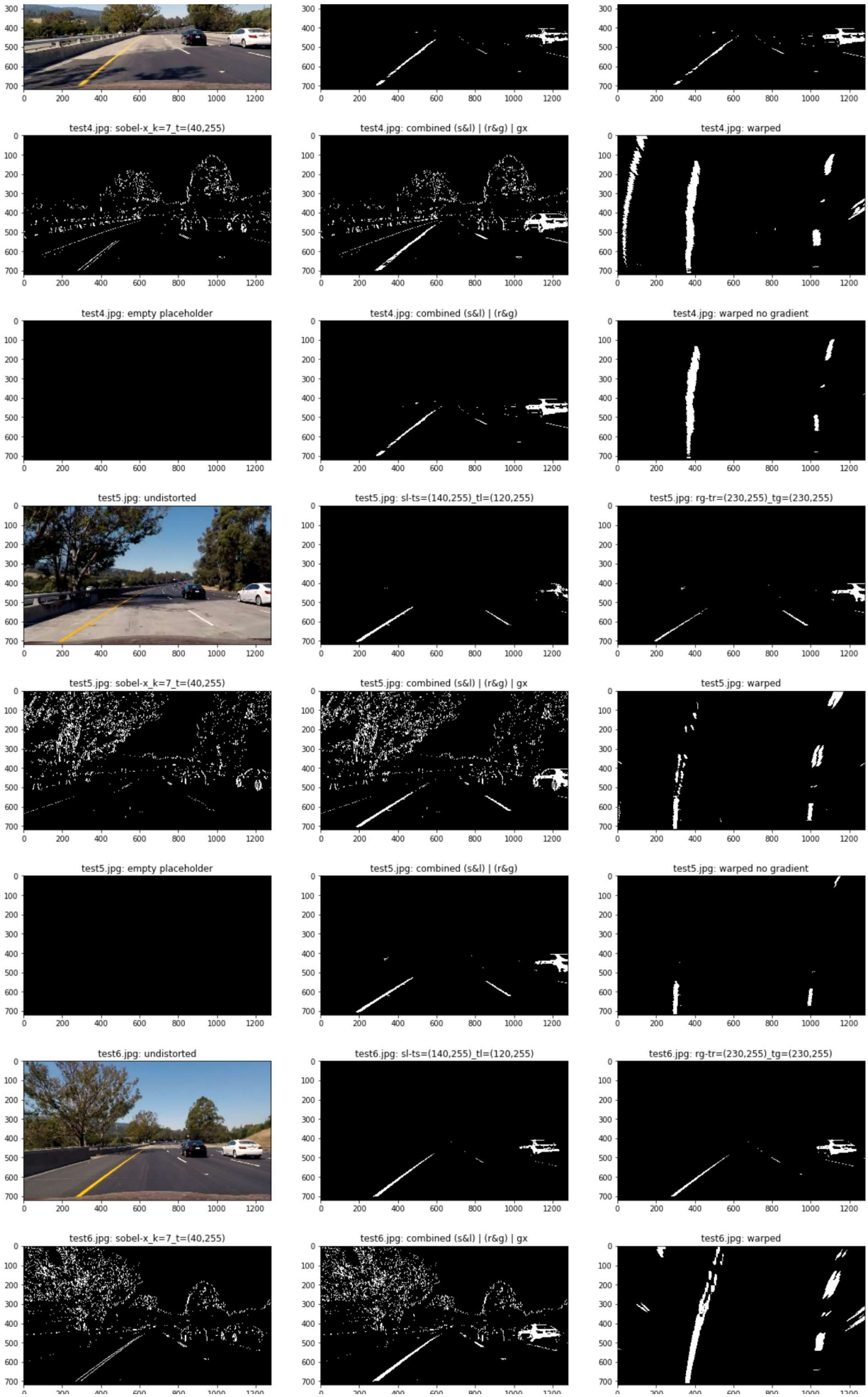
```
Loaded camera calibration
-------------------------
mtx:
 [[  1.15396093e+03   0.00000000e+00   6.69705359e+02]
 [  0.00000000e+00   1.14802495e+03   3.85656232e+02]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00]]

dist:
 [[ -2.41017968e-01  -5.30720497e-02  -1.15810318e-03  -1.28318543e-04
    2.67124302e-02]]
```

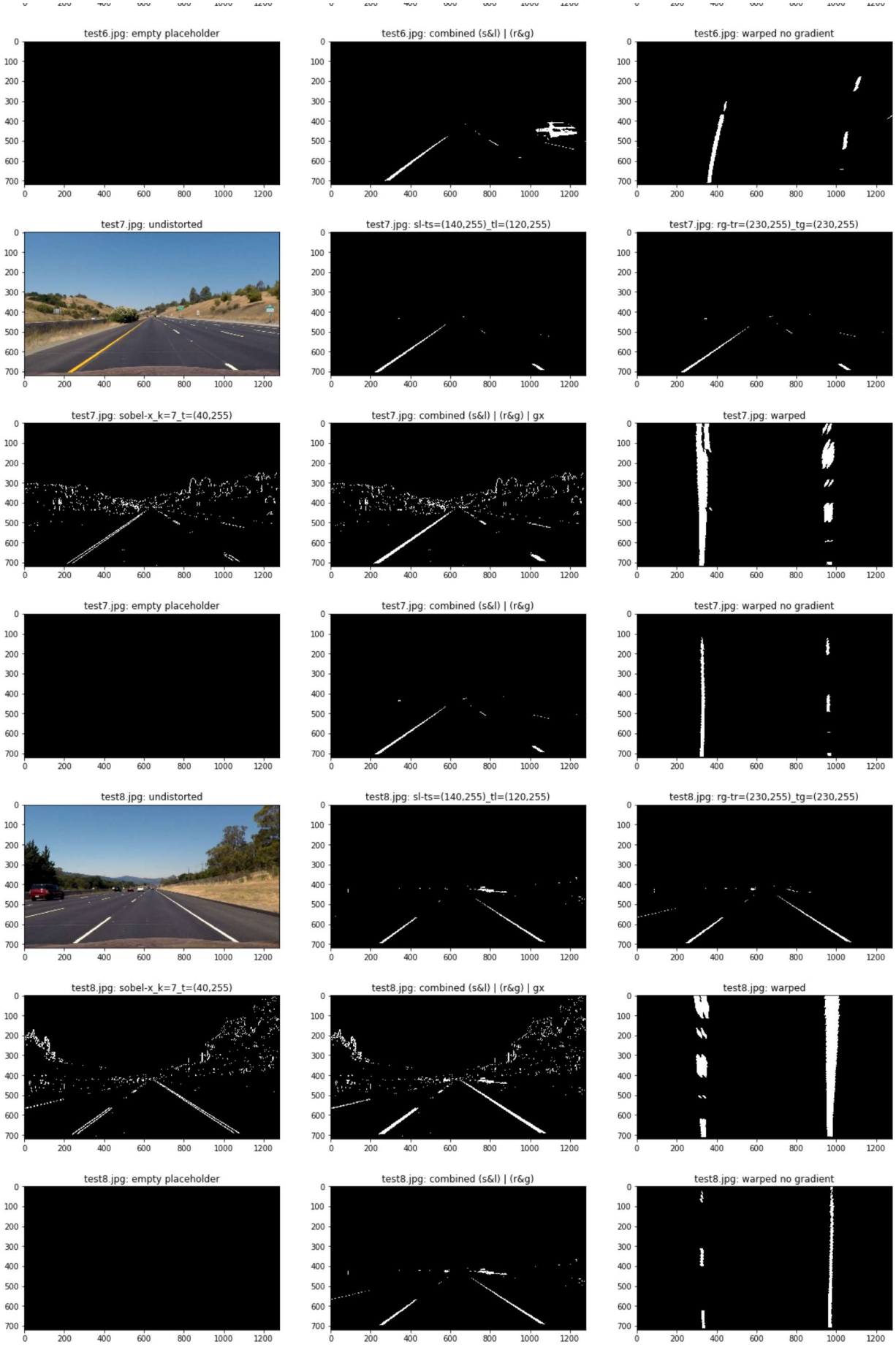
In [139]:
```
# abs sobel x (5, 25, 100)
# mag thresh (9, 30, 100)
# dir sobel (15, 0.8, 1.2)
# hls (100, 255)


#combined = np.zeros_like(dir_binary)
#combined[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1))
#color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, s_binary)

def process_image(img, img_name = ''):
    out = []
    out_n = []

    img_prefix = img_name + ': '

    #undistort image
    undistort = undistort_image(img)
    out.append(undistort)
    out_n.append(img_prefix + 'undistorted')

    # apply thresholds
    # s-channel will detect saturated colors like the white nd yellow from lane l
    binary_s = channel_select(undistort, channel=2, thresh=(140, 255))
    #out.append(binary_s)
    #out_n.append(img_prefix + 's-channel_t=(160,255)')

    # l-channel heps filtering out shadows
    binary_l = channel_select(undistort, channel=1, thresh=(120, 255))
    #out.append(binary_l)
    #out_n.append(img_prefix + 'l-channel_t=(0,120)')

    binary_sl = np.zeros_like(binary_s)
    binary_sl[(binary_s == 1) & (binary_l == 1)] = 1
    out.append(binary_sl)
    out_n.append(img_prefix + 'sl-ts=(140,255)_tl=(120,255)')

     # r and g channels help finding yellow and white
    binary_r = channel_select(undistort, channel=2, color_conversion=None, thresh
    #out.append(binary_r)
    #out_n.append(img_prefix + 'r-channel_t=(230,255)')

    binary_g = channel_select(undistort, channel=2, color_conversion=None, thresh
    #out.append(binary_g)
    #out_n.append(img_prefix + 'g-channel_t=(230,255)')

    binary_rg = np.zeros_like(binary_r)
    binary_rg[(binary_r == 1) & (binary_g == 1)] = 1
    out.append(binary_rg)
    out_n.append(img_prefix + 'rg-tr=(230,255)_tg=(230,255)')

    # detect vertical lines with sobel in x-direction
    binary_gx = abs_sobel_thresh(undistort, orient='x', sobel_kernel=7, thresh=(4
    out.append(binary_gx)
    out_n.append(img_prefix + 'sobel-x_k=7_t=(40,255)')
```

```
    # combine all threshold images
    binary = np.zeros_like(binary_s)
    binary[(binary_sl == 1) | (binary_rg == 1) | (binary_gx == 1)] = 1

    # draw line that was used for the perspective transform
    binary_lines = draw_lines(binary, warp_src)
    out.append(binary)
    out_n.append(img_prefix + 'combined (s&l) | (r&g) | gx')

    # warp image
    warped = warp_to_image_plane(binary)
    warped_lines = draw_lines(warped, warp_ln, skip = True)
    out.append(warped)
    out_n.append(img_prefix + 'warped')

    #put empty image to get array of 3x3
    out.append(np.zeros_like(binary_s))
    out_n.append(img_prefix + 'empty placeholder')

    # binary image based on color threshold only
    binary_no_grad = np.zeros_like(binary_s)
    binary_no_grad[(binary_sl == 1) | (binary_rg == 1)] = 1
    out.append(binary_no_grad)
    out_n.append(img_prefix + 'combined (s&l) | (r&g)')

    # warp image
    warped_no_grad = warp_to_image_plane(binary_no_grad)
    out.append(warped_no_grad)
    out_n.append(img_prefix + 'warped no gradient')


    return warped, out, out_n

out = []
out_n = []
i=4
for i in range(len(test)):
    warped, o, o_n = process_image(test[i], img_name = os.path.split(test_n[i])[-
    out += o
    out_n += o_n

plot_image_array(out, out_n, ncols=3)
```




test1.jpg: empty placeholder
test1.jpg: combined (s&l) | (r&g)
test1.jpg: warped no gradient
test2.jpg: undistorted
test2.jpg: sl-ts=(140,255)_tl=(120,255)
test2.jpg: rg-tr=(230,255)_tg=(230,255)
test2.jpg: sobel-x_k=7_t=(40,255)
test2.jpg: combined (s&l) | (r&g) | gx
test2.jpg: warped
test2.jpg: empty placeholder
test2.jpg: combined (s&l) | (r&g)
test2.jpg: warped no gradient
test3.jpg: undistorted
test3.jpg: sl-ts=(140,255)_tl=(120,255)
test3.jpg: rg-tr=(230,255)_tg=(230,255)
test3.jpg: sobel-x_k=7_t=(40,255)
test3.jpg: combined (s&l) | (r&g) | gx
test3.jpg: warped
test3.jpg: empty placeholder
test3.jpg: combined (s&l) | (r&g)
test3.jpg: warped no gradient
test4.jpg: undistorted
test4.jpg: sl-ts=(140,255)_tl=(120,255)
test4.jpg: rg-tr=(230,255)_tg=(230,255)


test4.jpg: sobel-x_k=7_t=(40,255)
test4.jpg: combined (s&l) | (r&g) | gx
test4.jpg: warped
test4.jpg: empty placeholder
test4.jpg: combined (s&l) | (r&g)
test4.jpg: warped no gradient
test5.jpg: undistorted
test5.jpg: sl-ts=(140,255)_tl=(120,255)
test5.jpg: rg-tr=(230,255)_tg=(230,255)
test5.jpg: sobel-x_k=7_t=(40,255)
test5.jpg: combined (s&l) | (r&g) | gx
test5.jpg: warped
test5.jpg: empty placeholder
test5.jpg: combined (s&l) | (r&g)
test5.jpg: warped no gradient
test6.jpg: undistorted
test6.jpg: sl-ts=(140,255)_tl=(120,255)
test6.jpg: rg-tr=(230,255)_tg=(230,255)
test6.jpg: sobel-x_k=7_t=(40,255)
test6.jpg: combined (s&l) | (r&g) | gx
test6.jpg: warped


test6.jpg: empty placeholder
test6.jpg: combined (s&l) | (r&g)
test6.jpg: warped no gradient
test7.jpg: undistorted
test7.jpg: sl-ts=(140,255)_tl=(120,255)
test7.jpg: rg-tr=(230,255)_tg=(230,255)
test7.jpg: sobel-x_k=7_t=(40,255)
test7.jpg: combined (s&l) | (r&g) | gx
test7.jpg: warped
test7.jpg: empty placeholder
test7.jpg: combined (s&l) | (r&g)
test7.jpg: warped no gradient
test8.jpg: undistorted
test8.jpg: sl-ts=(140,255)_tl=(120,255)
test8.jpg: rg-tr=(230,255)_tg=(230,255)
test8.jpg: sobel-x_k=7_t=(40,255)
test8.jpg: combined (s&l) | (r&g) | gx
test8.jpg: warped
test8.jpg: empty placeholder
test8.jpg: combined (s&l) | (r&g)
test8.jpg: warped no gradient