```
fi

for i in {1..4}
do
  case $1 in
  -d) shift; directory=$1; shift ;;
   *) url=${url:-$1}; shift;;
  esac
done

mkdir -p $directory;
baseurl=$(echo $url | egrep -o "https?://[a-z.]+")

echo Downloading $url
curl -s $url | egrep -o "<img src=[^>]*>" |
sed 's/<img src=\"\([^"]*\).*/\1/g' > /tmp/$$.list

sed -i "s|^/|$baseurl/|" /tmp/$$.list

cd $directory;

while read filename;
do
  echo Downloading $filename
  curl -s -O "$filename" --silent

done < /tmp/$$.list
```

An example usage is as follows:

```
$ ./img_downloader.sh http://www.flickr.com/search/?q=linux -d images
```

How it works...

The preceding image downloader script parses an HTML page, strips out all tags except `<img>`, then parses `src="URL"` from the `<img>` tag, and downloads them to the specified directory. This script accepts a web page URL and the destination directory path as command-line arguments. The `[ $# -ne 3 ]` statement checks whether the total number of arguments to the script is three, otherwise it exits and returns a usage example.

If there are three arguments, we parse the URL and destination directory. This is done as follows:

```
while [ -n "$1" ]
do
  case $1 in
  -d) shift; directory=$1; shift ;;
   *) url=${url:-$1}; shift;;
esac
done
```

A `while` loop is used. It runs as long as there are more arguments to be processed. The `case` statement will evaluate the first argument (`$1`), and `matches -d` or any other string arguments are checked. The advantage of parsing arguments in this way is that we can place the `-d` argument anywhere in the command line:

```
$ ./img_downloader.sh -d DIR URL
```

Or:

```
$ ./img_downloader.sh URL -d DIR
```

`shift` is used to shift arguments to the left in such a way that when shift is called, `$1` will take the next argument's value; that is, `$2`, and so on. Hence, we can evaluate all arguments through `$1` itself.

When `-d` is matched, it is obvious that the next argument is the value for the destination directory. `*)` corresponds to a default match. It will match anything other than `-d`. Hence, while iteration `$1=""` or `$1=URL` in the default match, we need to take `$1=URL`, avoiding `""` to overwrite. Hence, we use the expression `url=${url:-$1}`. It will return a URL value if already not `""`, otherwise it will assign `$1`.

`egrep -o "<img src=[^>]*>"` will print only the matching strings, which are the `<img>` tags including their attributes. `[^>]*` is used to match all the characters except the closing `>`, that is, `<img src="image.jpg" …. >`.

`sed 's/<img src=\"\([^"]*\).*/\1/g'` parses `src="url"`, so that all the image URLs can be parsed from the `<img>` tags already parsed.

There are two types of image source paths—relative and absolute. **Absolute paths** contain full URLs that start with `http://` or `https://`. Relative URLs starts with `/` or `image_name` itself. An example of an absolute URL is `http://example.com/image.jpg`. An example of a relative URL is `/image.jpg`.