inango
SOFTWARE VIRTUOSITY

Yocto Source Mirrors Mechanism

By Nikolay Merinov, Software Team Leader

December 28, 2018

Foreword

The BitBake build utility, adapted by the Yocto Project, allows for flexible configuration of source mirrors for all the remote files required during the build of Yocto distribution. This functionality allows to speed up the distribution build, as well as organize a local backup with sources of all the packages used.

However, this mechanism is rather poorly documented, and we are going to fix this situation in the current article.

BitBake Source Mirrors

The BitBake mirrors mechanism is supported at least in three places:

1. `PREMIRRORS` are the primary mirrors that will be used before the upstream URI from `SRC_URI`. It can be used to provide quick mirrors to speed-up the sources downloading.
2. `MIRRORS` are the additional mirrors that are used when an upstream URI from `SRC_URI` is not accessible. Creating `MIRRORS` is a good idea for a long-living distributions, when a distribution can out-live the upstream sources of a used software.
3. `SSTATE_MIRRORS` are the mirrors for a prebuilt cache data objects. `SSTATE_MIRRORS` can be used to share the pre-built objects from a CI builds.

The Yocto Project additionally provides the support for the next mechanisms:

1. `own-mirrors.bbclass` is a standard way to provide a single pre-mirror for all the supported fetchers:

   ```
   INHERIT += "own-mirrors"
   SOURCE_MIRROR_URL = "http://example.com/my-source-mirror"
   ```

2. `BB_GENERATE_MIRROR_TARBALLS` variable can be set to "1" to allow reusing `DL_DIR` from the current build as a mirror for other builds. Without a `BB_GENERATE_MIRROR_TARBALLS` variable the resulting `DL_DIR` would not provide suitable mirrors for sources fetched from VCS repositories.
3. `BB_FETCH_PREMIRRORONLY` and `SSTATE_MIRROR_ALLOW_NETWORK` variables can be used to configure BitBake to download sources and build artifacts only from the configured mirrors. These variables will be usable for all the `BB_NO_NETWORK` configurations.

Yocto Documentation on Mirrors Syntax

All things described in the previous part are correctly documented and can be found in the official documentation. But a syntax for mirror rules itself is poorly documented. An official documentation suggests only three examples how mirror rules can be created:

1. In the example[1] for a `PREMIRRORS` variable we can see that we should use `.*/.*` regexp to match all URI for a specific fetcher:

```
PREMIRRORS_prepend = "\
git://.*/.* http://www.Yoctoproject.org/sources/ \n \
ftp://.*/.* http://www.Yoctoproject.org/sources/ \n \
http://.*/.* http://www.Yoctoproject.org/sources/ \n \
https://.*/.* http://www.Yoctoproject.org/sources/ \n"
```

2. In the example[2] for a `SSTATE_MIRRORS` variable we can find that there is also a support for a `PATH` string in the matching:

```
SSTATE_MIRRORS ?= "\
file://.* http://someserver.tld/share/sstate/PATH;downloadfilename=PATH
\n \
file://.* file:///some-local-dir/sstate/PATH"
```

3. In the same `SSTATE_MIRROS` variable from example[2] we can additionally find an information that a mirror syntax supports a more sophisticated regex usage:

```
SSTATE_MIRRORS ?= "file://universal-4.9/(.*)
http://server_url_sstate_path/universal-4.8/\1 \n"
```

But there is no full description of the mirror matching mechanism. We offer the following as a way of filling this gap.

Yocto Mirror Syntax

All mirrors are defined as a "pattern replacement" pair. The BitBake attempts to match a source URI against each "pattern" and on a successful match the BitBake generates a new mirror URI based on an original URI, by using the "pattern" and "replacement" definitions.

URI matching

Each URI, including the "pattern" and "replacement" strings, is split into six components before matching as follows:
`scheme://user:password@hostname/path;parameters.`

In an upstream URI this parts is understood as strings, in a "pattern" this parts is interpreted as a Python `re` regexps (`parameters` is the only field that is not regexp); in a "replacement" this parts is interpreted as regexp replacements in terms of a Python `re.sub` function.

All parts from an upstream URI are matched against regexps from the corresponding part of a "pattern" with the next additional rules:

1. The BitBake always adds a `$` sign to the end of a `scheme` regex. It was done to ensure that the "http" `scheme` would not match the "https" string.
2. If a `scheme` is a "file" then the `hostname` is assumed to be empty. Otherwise the `hostname` is a text between "://" and next "/" characters.
3. `parameters` are interpreted as a list of "param=value" pairs. For each such pair specified in a "pattern" there should be a full match in an original `SRC_URI`. It is the only field that is matched with a string equality instead of a regexp matching.
   NOTE: A parameters matching was broken up to Yocto 2.6 release. This issue was found and fixed as a part of work on this document.

URI Replacement

If a URI was successfully matched, then the BitBake creates a new mirror URI from an upstream URI using a "replacement" string. The BitBake uses the next rules to make a replacement:

1. A replacement is made for each part of the URI separately.
2. Before replacement is performed, we replace the next substrings in a corresponding part of a "replacement" string:
   1. "TYPE" -- this substring in a "replacement" string will be replaced with an original URI scheme.
   2. "HOST" -- will be replaced with an original URI hostname.
   3. "PATH" -- will be replaced with an original URI path.
   4. "BASENAME" -- will be replaced with a basename of a "PATH" (with a substring after the last "/" symbol).
   5. "MIRRORNAME" -- will be replaced with a special string obtained as concatenation of a "HOST" with `":"` symbols replaced with `"."` symbols, and a "PATH" with `"/"` and `"*"` symbols replaced with a `"."` symbols.

   Note: In the `parameters` list this replacement is made only in a `value` part of the `param=value` pairs.

3. Each part except the `parameters` is replaced with `re.sub(part_from_pattern, part_from_replacement, part_from_original_uri, 1)` Python command. This means that the BitBake will replace only a first match from a current part and that you can use a Python regex replacement syntax including `"\1"` syntax to insert parts of a matched URI to a result.
4. If an original URI `scheme` differs from a "replacement" `scheme` then the original `parameters` will be wiped off.

5. The `parameters` from a "replacement" should be added to a resulting URI. If there were parameters with the same names in an original URI then the value for these parameters will be overridden.
6. Finally, the BitBake checks that a `path` part upon replacement is finished with a suggested "basename". If a resulting `path` ends with any other string then a resulting `path` will be concatenated with a `"/basename"` string. A suggested "basename" is created according to the next rules:
    1. If a `scheme` was not changed then the "basename" is just a basename of the `path` from an original URI.
    2. If a `scheme` was changed and an original URI points to a single file or a tarball with sources, then the same `basename` will be used.
    3. If a `scheme` was changed and an original URI points to some kind of a repository then the `mirrortarball` name will be used. This name is a fetcher specific. e.g. for a git repository a tarball name will be "git2_hostname.path.to.repo.git.tar.gz".

NOTE: The last step means that it is impossible to use the `"file:///some/path/b.tar.gz"` as a mirror path for the `"http://other/path/a.tar.gz"`, but you still can use the `"file:///some/path/a.tar.gz"` or the `"file:///some/path/prefix_a.tar.gz"`.

Step by Step Process Description

Let's check how the next example will be processed:

```
SRC_URI = "git://git.invalid.infradead.org/foo/mtd-
utils.git;branch=master;tag=1234567890123456789012345678901234567890"
PREMIRRORS = "git://.*/.*;branch=master
git://somewhere.org/somedir/MIRRORNAME;protocol=http;branch=master_backup \n"
```

First of all, we will split all the three URI's into the next parts:

```
upstream:
    scheme     = "git"
    user       = ""
    password   = ""
    host       = "git.invalid.infradead.org"
    path       = "/foo/mtd-utils.git"
    parameters = {"branch": "master", "tag":
"1234567890123456789012345678901234567890"}
pattern:
    scheme     = "git$"
    user       = ""
    password   = ""
    host       = ".*"
    path       = "/.*"
    parameters = {"branch": "master"}
replacement:
    scheme     = "git"
    user       = ""
    password   = ""
```

```
    host       = "somewhere.org"
    path       = "/somedir/MIRRORNAME"
    parameters = {"branch": "master_backup", "protocol": "http"}
```

On the next step the BitBake will match the "upstream" URI parts against the corresponding parts of a "pattern" with a `re.match(pattern.part, upstream.part)` command for all the parts except the parameters. For the "parameters" the BitBake will check that the value for a "branch" parameter in an "upstrem" URI and a "pattern" URI are equal.

When these checks pass, the BitBake will start a replacement process. In each part of the "replacement" BitBake will make the replacements for the special strings:

```
replacement:
    scheme     = "git"
    user       = ""
    password   = ""
    host       = "somewhere.org"
    path       = "/somedir/git.invalid.infradead.org.foo.mtd-utils.git"
    parameters = {"protocol": "http", "branch": "master_backup"}
```

Then the BitBake will make the actual replacements with a `re.sub(pattern.part, replacement.part, upstream.part, 1)` command for all parts except the `parameters`. This means that if you want to replace the string completely, the full pattern should be a `".*"`, and not `""`. For the `parameters` the new keys will be added to the list and the old values will be replaced. As the result we will get as follows:

```
result:
    scheme     = "git"
    user       = ""
    password   = ""
    host       = "somewhere.org"
    path       = "/somedir/git.invalid.infradead.org.foo.mtd-utils.git"
    parameters = {"branch": "master_backup", "protocol": "http", "tag":
"1234567890123456789012345678901234567890"}
```

On the last step the BitBake will check that the `result.path` is finished with a `basename` of an `upstream.path`. Since the `result.scheme` and the `upstream.scheme` are the same, the `basename` will be defined as an `mtd-utils.git` (if a scheme will be different, then the `basename` would be defined to a `"git2_git.invalid.infradead.org.foo.mtd-utils.git.tar.gz"`).

When the last check passes, the BitBake will combine a result to a new mirror URI that will be used as an alternative source for the files:
`git://somewhere.org/somedir/git.invalid.infradead.org.foo.mtd-utils.git;branch=master_backup;protocol=http;tag=1234567890123456789012345678901234567890`.

You can find additional replacements examples in the BitBake unit tests code[3].

Used Documentation

[1]: https://www.Yoctoproject.org/docs/current/ref-manual/ref-manual.html#var-PREMIRRORS 'Simple example'

[2]: https://www.Yoctoproject.org/docs/current/mega-manual/mega-manual.html#var-SSTATE_MIRRORS 'extended example'

[3]: https://github.com/openembedded/BitBake/blob/master/lib/bb/tests/fetch.py#L374 'Additional replacement examples'

[4]: https://github.com/openembedded/BitBake/blob/master/lib/bb/fetch2/__init__.py#L904 'BitBake mirror mechanism implementation'