Git 2.14.4 for z/OS Release Notes

This document provides tips for effectively using the Rocket Software port of Git 2.14.4 to z/OS. The reader should already have some familiarity with Git, z/OS, and the Unix System Services component of z/OS.

z/OS-specific changes since the previous (2.3.5) release

In addition to the platform-independent changes between Git release 2.3.5 and 2.14.4, the following changes apply to the z/OS port.

- The `https` protocol is now supported for access to remote repositories, in addition to the `ssh` protocol.
- The `working-tree-encoding` attribute has been replaced by a new `zos-working-tree-encoding` attribute.

> It is vital that all users of the `working-tree-encoding` attribute migrate to the new `zos-working-tree-encoding` attribute as soon as possible. See the notes below regarding an incompatibility with git releases after 2.18 on other platforms.

Background

Git is currently the dominant source code control system used by the open source software community. Its first and most well-supported platform is Linux, running on Intel x86 hardware. An underlying assumption has been that the source code being managed by Git will be encoded in a character encoding closely related to ASCII, typically either ISO8859-1 or UTF-8. Git does not alter the encoding between the repository itself (the "repo") and the checked-out "working tree"; if a project's files are encoded with a mixture of encodings, the build system for the project is presumed to be able to manage that mix.

z/OS and its predecessors (OS/360, MVS and OS/390), however, use EBCDIC, most commonly the IBM-1047 variant, as their primary character encoding. When Unix System Services (USS) was introduced, it continued that practice. The single most difficult challenge when porting open source software projects to USS is managing this encoding difference. This generally shows up in two ways:

- The project's source code must be converted from its native encoding to EBCDIC prior to attempting to build it.
- The project's code might have embedded presumptions, regarding the runtime character encoding, that do not hold true for EBCDIC.

Beginning with z/OS Version 1 Release 2, IBM introduced a collection of features known as enhanced ASCII support that are intended to solve these problems. These features include:

- the ability to "tag" a file with the encoding used by that file
- C/C++ compiler support for using ASCII at runtime as a program's internal character encoding
- automatic conversion of data between the encoding in a file and the encoding used internally by a program
- runtime services for manipulating file tags and controlling data conversion
- support in many commonly-used Unix commands for ASCII data, including respect for file tags and explicit control over data conversion

Git for z/OS uses these features to enable the user to:

- use open source projects' Git repositories (such as on github or savannah), including "pushing" changes back to those repositories
- choose the encoding of source files in the working tree

The remainder of this document describes:

- the initial setup steps to ensure that Git functions correctly on z/OS
- recommended approaches to using Git on z/OS

Setup

Installation directory

Git and the prerequisite software listed below must be installed into a directory that is accessible to all users who will be using Git. This may be the same directory as is used for other Rocket-ported open source tools.

The instructions below presume that the name of this directory has been set in the environment variable RSUSR.

Prerequisite software

Git for z/OS requires the latest Rocket z/OS ports of:

- `bash 4.3`
- `perl 5.24`
- `unzip 6.0`

The installation instructions in `$RSUSR/share/doc/*/README.ZOS` must be followed for each tool. Once installation is complete, the installation directory will look something like this (just the top 2 levels, with many files elided):

```
bin/
    bash
    git
    perl
    unzip
    ...and much more...
lib/
    perl5
libexec/
    git-core
man/
    cat1
    cat5
    cat7
share/
     doc
     git-core
     man
```

Downloading a certificate bundle

The z/OS port of Git 2.14.4 now supports the `https` Git remote access protocol in addition to the `ssh` protocol. To use this, you must set the environment variable `GIT_SSL_CAINFO` to point to a file containing the X.509 certificates of the public Certificate Authorities, in PEM format.

If you do not already have a suitable certificate file, you can download a current copy of the file from a trusted source and verify the signature of the file. A suggested source is the `curl` web site. If you have the Rocket ports of `curl` and `openssl` installed, you can use the following commands. These assume that:

- The path of the directory in which Git and the related tools were installed is in the environment variable RSUSR.
- You have write permission to that directory.

- You wish to store the certificate file in that directory.

```
# Make sure that there is an "etc" subdirectory in the Rocket ported tools
directory
mkdir -p $RSUSR/etc
cd $RSUSR/etc

# Get the certificate file
curl -s -k https://curl.haxx.se/ca/cacert.pem -o cacert.pem

# Get the signature file and extract just the hash
curl -s -k https://curl.haxx.se/ca/cacert.pem.sha256 | awk ' {print $1}' >
cacert.pem.sha256

# Generate the hash on the certificate file and compare it to the signature
file.
# If the signature matches, there will be no output from diff.
openssl dgst -sha256 cacert.pem | awk ' {print $2}' | diff -
cacert.pem.sha256
```

Once this has been done, you can set `GIT_SSL_CAINFO` to point to the file:

```
export GIT_SSL_CAINFO=$RSUSR/etc/cacert.pem
```

It is also possible for a Git user to disable the certificate checking by entering the following command. This is not recommended.

```
git config --global http.sslVerify false
```

Environment variable settings

This script sets the appropriate values after RSUSR has been correctly set.

Sample shell code (for .bashrc)

```
# This must be sourced, not run

# Replace this value with your installation directory. The value is ONLY
# used in this script to set other values.
export RSUSR=/u/yourid/git

export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS='FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)'
export _TAG_REDIR_IN=txt
export _TAG_REDIR_OUT=txt
export _TAG_REDIR_ERR=txt

export PATH=$RSUSR/bin:$PATH
export LIBPATH=$RSUSR/lib/perl5/5.24.0/os390/CORE:$LIBPATH
export PERL5LIB=$RSUSR/lib/perl5
export MANPATH=$RSUSR/man:$MANPATH

export GIT_SHELL=$RSUSR/bin/bash
export GIT_EXEC_PATH=$RSUSR/libexec/git-core
export GIT_TEMPLATE_DIR=$RSUSR/share/git-core/templates
export GIT_MAN_PATH=$RSUSR/man


# To use the https protocol, Git requires a certificate bundle.
export GIT_SSL_CAINFO=$RSUSR/etc/cacert.pem

# OPTIONAL: Set the default editor used by git for commit messages to be
vi, and
#          to encode the comments in ISO8859-1
# git config --global core.editor "/bin/vi -W filecodeset=ISO8859-1"
```

Using Git

Git for z/OS introduces two new concepts into Git:

- `git-encoding`: The character encoding of files stored in the Git repository. These files are part of Git's internal database and are never directly edited by users.
- `zos-working-tree-encoding`: the character encoding of files stored in the working directory. These files are the reason Git exists. They are the user-defined content of the repository, and are what the user edits and compiles.

These two encodings are controlled via the Git attribute system.

If no attributes are set, Git for z/OS behaves essentially the same as it does on all other platforms: the encoding is never altered when files are copied between the working directory and the repository. However, there is one additional aspect of z/OS-specific behavior: when files are created in the working directory (via, for example, Git checkout), they are tagged as being encoded in ISO8859-1 (ASCII). These tags can be examined by using either the `ls` command with the `-T` option or the `chtag` command with the `-p` option. The output of those commands in a directory in which file tags have been set is as follows:

```
bash-2.03$ ls -lT
total 1000
t ISO8859-1   T=on  -rw-r--r--   1 TSJLC    PDUSER      3240 Oct 20 14:22
Makefile
t ISO8859-1   T=on  -rw-r--r--   1 TSJLC    PDUSER       144 Oct 20 14:22
README.md
- untagged    T=off -rwxr-xr-x   1 TSJLC    PDUSER    372736 Oct 20 15:38
sysevent
t ISO8859-1   T=on  -rw-r--r--   1 TSJLC    PDUSER      1224 Oct 20 14:22
sysevent.cpp
m IBM-1047    T=off -rw-r--r--   1 TSJLC    PDUSER    126000 Oct 20 15:38
sysevent.o
bash-2.03$ chtag -p *
t ISO8859-1   T=on  Makefile
t ISO8859-1   T=on  README.md
- untagged    T=off sysevent
t ISO8859-1   T=on  sysevent.cpp
m IBM-1047    T=off sysevent.o
bash-2.03$
```

Most z/OS commands that manipulate text files (editors, such as `vi` and `emacs` 19.34, the XL C and C++ compilers, `make, grep,` etc.) will respect the tag, if present, and "do the right thing" with the text. If a file is not tagged, it is generally presumed to be encoded in IBM-1047 (EBCDIC).

This default behavior presumes that the files in the Git repository are encoded in ISO8859-1.

The encoding tag on a file is changed with using the chtag command, like this:

```
chtag -t -c iso8859-1 <filename>   # to tag as ASCII
chtag -t -c ibm-1047 <filename>    # to tag as EBCDIC
chtag -t -c utf-8 <filename>       # to tag as UTF-8
```

Supported encodings

There are three character encodings that will work well in this system:

- ISO8859-1 (ASCII)
- IBM-1047 (EBCDIC)
- UTF-8

Binary (non-textual) files are also supported.

A crucial characteristic of the ISO8859-1/IBM-1047 pair is that every one of the 256 characters in each of these encodings maps uniquely to a character in the other. This means that a file can be converted from one to the other and back with no loss of information.

In addition, every character in ISO8859-1 and IBM-1047 has a unique mapping to UTF-8. This means a file can be converted from either ISO8859-1 or IBM-1047 to UTF-8 and back with no loss of information. The reverse is not true; there are many characters that can be encoded in UTF-8 that have no mapping in ISO8859-1 or IBM-1047. This obviously must be true, since ISO8859-1 and IBM-1047 represent each character in a single byte, which has only 256 possible values, while UTF-8 is capable of encoding over a million characters (in one to four bytes).

Other IBM-xxxx encodings will work mostly if _BPXK_AUTOCVT is set to ALL. However, due to the nature of the character sets, full bi-directional mapping is not supported.

The following table provides recommended combinations of `git-encoding` and `zos-working-tree-encoding`. Note: it is reasonable for different files to use different combinations, depending on the content.

Default value: git-encoding=ISO8859-1 zos-working-tree-encoding=ISO8859-1

git-encoding	zos-working-tree-encoding	Usage
unspecified	unspecified	This is the default behavior: iso8859-1 is presumed for both the repository and the working directory. The file in the working directory will be encoded and tagged as ISO8859-1.
ISO8859-1	ISO8859-1	The same as the default behavior, but explicitly specified.
ISO8859-1	IBM-1047	This causes the file in the working directory to be encoded and tagged as IBM-1047. This allows EBCDIC-only tools to be used with the file.
UTF-8	UTF-8	This is useful for files that contain characters that cannot be represented in a single byte, or for files that will be processed as UTF-8 (such as web content).
unspecified	IBM-1047	This causes the file in the working directory to be encoded and tagged as IBM-1047. This allows EBCDIC-only tools to be used with the file.
unspecified	ISO8859-1	The same as the default behavior, but explicitly specified.

Changing the encodings

Note: Avoid changing encodings if possible!

As noted above, most z/OS tools (editors, compilers, shell commands, etc.) can work directly with files encoded and properly tagged as ASCII and do not require source code to be encoded as EBCDIC. Git for z/OS will automatically provide the correct file tagging.

This means that the `.gitattribute` support provided by Git for z/OS is often not necessary. If the project you are working on can avoid using the `zos-working-tree-encoding` attribute, you should do so and work with your files in ASCII.

As noted, the Git attribute system is used to specify the encoding of files in both the Git repository and the working directory. Attributes are set by creating an attribute specification, which consists of a file-matching pattern (as specified when creating a .gitignore file) and one or more attributes. The attribute specification can reside in either the working directory (in a `.gitattributes` file) or in the Git repository (usually named `.git`), in the file `info/attributes`.

When stored in the working directory, the file can be committed to the repository like any other file, in which case the attribute specification will be automatically applied when the working directory is created via `git checkout`.

Some projects might not want to introduce a z/OS-specific file into the repository. In these cases, the attributes can be stored in the `.git/info/attributes` file.

Note: Attribute specifications, whether in `.gitattributes` or `.git/info/attributes`, must be encoded in ISO8859-1, and must be tagged as such.

A typical Git attribute file might look something like this:

```
*.md zos-working-tree-encoding=ibm-1047 git-encoding=iso8859-1
*.cpp zos-working-tree-encoding=ibm-1047 git-encoding=iso8859-1
Makefile zos-working-tree-encoding=ibm-1047 git-encoding=iso8859-1
*.png binary
```

Note: General usage specification of Git attributes files can be found here and specific documentation of binary option can be found here
.

A Git attribute specification can be tested by using the `git check-attr` command:

```
bash-2.03$ git check-attr -a *
Makefile: zos-working-tree-encoding: ibm-1047
Makefile: git-encoding: iso8859-1
README.md: zos-working-tree-encoding: ibm-1047
README.md: git-encoding: iso8859-1
sysevent.cpp: zos-working-tree-encoding: ibm-1047
sysevent.cpp: git-encoding: iso8859-1
*.png: binary: set
*.png: diff: unset
*.png: merge: unset
*.png: text: unset
bash-2.03$
```

This `.gitattributes` files will be suitable for many projects:

```
# This .gitattributes file will cause all text files EXCEPT for
# git's .gitattributes and .gitignore files to be encoded as EBCDIC.
# Selected binary files will not be translated at all.

# The default for text files
*       git-encoding=iso8859-1 zos-working-tree-encoding=ibm-1047

# git's files (which MUST be ASCII)
.gitattributes   git-encoding=iso8859-1 zos-working-tree-encoding=iso8859-1
.gitignore       git-encoding=iso8859-1 zos-working-tree-encoding=iso8859-1

# Binary files
*.jpg   binary
*.png   binary
*.gif   binary
*.zip   binary
```

Cloning an open source repository

The remote access protocols currently supported by Git for z/OS are `ssh` and `https`.

- To use `ssh`, you will need ssh access credentials on the remote server. In the case of github, accounts are available for free, and publicly-readable repositories do not require any further authorization to be cloneable.
- To use `https`, you generally do not need credential on the remote server. However, you do need credentials to push code to the remote server.

Here is an example of cloning a repository from github. If you have an account on github, this should work once you have installed Git for z/OS and configured the environment correctly.

```
bash-2.03$ git clone  -v git@github.com:zorts/hello_world.git
Cloning into 'hello_world'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 3), reused 13 (delta 3), pack-reused 0
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (3/3), done.
Checking connectivity... done.
bash-2.03$
```

This particular repository has a `.gitattributes` file in its top-level directory that will arrange for the files to be checked out as EBCDIC.

Restrictions

- Some files used by Git must be encoded and tagged as ISO8859-1. These include:
  - Git attribute files, whether in `.git/info/attributes` or `.gitattributes`
  - `.gitignore` files
- The only remote protocols supported are `ssh` and `https`.
- Only client mode is supported; in other words, Git for z/OS can clone from, and push to, remote repositories via `ssh or https`, but cannot be the target of clone and push from other clients.
- The only encodings supported for `git-encoding` are ISO8859-1 and UTF-8.
- The only encodings fully supported for `zos-working-tree-encoding are ISO8859-1`, `IBM-1047 and UTF-8.`

Migrating from working-tree-encoding to zos-working-tree-encoding

The previous release of git for z/OS (version 2.3.5) used `working-tree-encoding`, rather than `zos-working-tree-encoding` for controlling the encoding of files in the working tree. It is vital that such usage be migrated as soon as possible to the new attribute name.

Why is this necessary?

Release 2.18 of the platform-generic git code introduced a git attribute with the same name (`working-tree-encoding)` as the one used by the Rocket z/OS port of release 2.3.5. The meaning is essentially the same; however, this means that attempts to use a repository prepared on z/OS with other open source platforms (such as Windows, or the servers that typically run git servers such as GitHub or BitBucket) will now attempt to process this attribute, almost certainly with undesired results. It was considered unlikely that this attribute would ever appear in the platform-generic git; this turned out to be wrong.

What action is required?

Git repositories on z/OS that use the `working-tree-encoding` attribute should be altered immediately to use the new `zos-working-tree-encoding` attribute.

The `working-tree-encoding` attribute will continue to be honored on z/OS; however, as time goes on and git on other platforms is updated to 2.18 and beyond, the likelyhood of problems when using GitHub, BitBucket, and Windows git-based tools will increase.

References

- Background discussion on why working-tree-encoding was added to the platform-generic code
- The first report of the conflict involving the z/OS attribute

Known issues

- The installation directory for `PATH`, `MANPATH`, `INFO`, and `HTML` must be manually changed from the default (`/rsusr/rocket`) to the Git installation directory of your choice. Environment variables must be updated accordingly.
- Very large repositories (especially repositories with a lot of history) may cause clone and checkout operations to fail with this

symptom:

```
 fatal: Out of memory? mmap failed: EDC5124I Too many open files. (errno2=0x07360344)
```

The solution is to restrict Git to using less memory, by setting these configuration variables:

```
pack.packSizeLimit 20m
core.packedGitWindowSize 16m
core.packedGitLimit 32m
pack.windowMemory 32m
pack.thread 1
pack.deltaCacheSize 1m
```