perm.pub/dsi:VGajCjaNP1Ugz58Khn1JWOEdMZ8/1.1
Additional formats and editions available online.

Edition 1.1

Author date: 2024-02-20

Archive date: 2024-02-21

Citation:
E. Castedo Ellerman (2024) "Document Succession Git Layout" perm.pub https://perm.pub/dsi:VGajCjaNP1Ugz58Khn1JWOEdMZ8/1.1

Document Succession Git Layout

E. Castedo Ellerman (castedo@castedo.com)

Abstract

DOCUMENT TYPE: Living Technical Specification

A document succession is a distributed, correctable document that preserves document snapshots as editions, which can be individually referenced. The Document Succession Git Layout (DSGL) is a format for storing document successions within a Git object store. DSGL bridges two related formats: the textual representation of Document Succession Identifiers (DSI) and the storage format for document snapshots. An example of a snapshot format is the format for Baseprint document snapshots; however, this specification does not define any specific format for document snapshots.

Background

Websites like https://perm.pub use free open-source software, such as the Python package Epijats, to process the formats of Document Succession Git Layout (DSGL), Document Succession Identifiers (DSI), and Baseprint document snapshots. For the motivation behind these technologies, refer to Why Publish Baseprint Document Successions. Tutorials and introductory materials are also available at https://try.perm.pub/.

Scope

This document is a specification of DSGL for interoperability with the following reference implementations:

- the Python package Hidos version 1.3 [1] and
- the Document Succession Highly Manual Toolkit [2].

This specification excludes potential DSGL features that are not implemented in any software. The online forum https://baseprints.singlesource.pub is available for proposals of improvements to this living specification and its reference implementations.

DSI in this specification refers to edition 2 of the Document Succession Identifier specification.

Signed Successions

This specification focuses on document successions that are digitally signed. *Unsigned* document successions fall outside the scope of this edition of the DSGL specification. While unsigned document successions may be useful for testing and learning, they are not essential for interoperability.

Ungarbled Successions

This specification defines DSGL for *ungarbled* recordings. These *ungarbled* recordings, which have a simple and intuitive format, are most likely to interoperate. Reality and non-ideal circumstances sometimes result in *garbled* recordings, which software may handle with varying degrees of success. This specification assumes document successions are *ungarbled*, unless otherwise noted.

Informal Description

Initial Git Commit

A document succession consists of a Git commit history with a single initial commit. The base DSI of the document succession is the base64url (RFC 4648)[3] representation of the 20-byte binary Git hash of the initial commit. An *ungarbled* document succession has a linear Git commit history.

Signatures

Every Git commit tree in DSGL contains a `signed_succession` subdirectory that includes an `allowed_signers` file listing the public keys allowed to extend the document succession. Each non-initial commit is signed with an SSH key listed in the `allowed_signers` files of all its parent commits.

Example: Initial Git commit in DSGL.

> *https://github.com/document-succession/1wFGhvmv8XZfPx0O5Hya2e9AyXo/commit/d7014686f9aff1765f3f1d0ee47c9ad9ef40c97a*

> *https://archive.softwareheritage.org/swh:1:rev:d7014686f9aff1765f3f1d0ee47c9ad9ef40c97a*

Document Snapshot

A document snapshot in DSGL is a digital object that can be encoded as either a Git blob or a Git tree. The contents of each document snapshot are stored in an entry named `object` within a containing Git tree. The full path from the Git commit tree mirrors the edition number, with slashes separating integers instead of periods. For example, the directory containing the contents of a document snapshot for edition 2.1 would be stored in a Git tree at the path `2/1/object`.

Example: Git commit tree of document succession with edition 2.1.

> *https://github.com/document-succession/1wFGhvmv8XZfPx0O5Hya2e9AyXo/tree/f174a4f4cc3076b0f46980878c4208cbfcdb990b*

> *https://archive.softwareheritage.org/swh:1:dir:361534f2bcf78e6312a32916469e4720c7c9bb6f*

Formal Definitions

Criteria for a Document Succession in DSGL

Criterion: The data of a document succession are fully recorded by a connected graph of Git commit records.

Criterion: There is only one initial Git commit (a commit without parents) in the graph of Git commit records. The 20-byte binary hash of this initial Git commit is the hash used for the base DSI of the document succession.

Criterion: All document snapshots are stored as a Git blob or Git tree with the name `object` (in its containing Git tree).

Criterion: The Git tree containing an `object` entry is not the top-level Git commit tree and is named with a positive integer.

Criterion: The full path to the containing tree of an `object` entry for a document snapshot consists of non-negative integers separated by slashes. These non-negative integers correspond to the non-negative integers separated by periods in the DSI.

Criterion: The document snapshot assigned to an edition number is the first Git blob or tree committed to an `object` entry following the path corresponding to the edition number. Any subsequent `object` entries at this path do not change the DSI assignment.

Criteria for a Signed Document Succession in DSGL

Criterion: The Git tree of every Git commit record has an `allowed_signers` file within the `signed_succession` directory.

Criterion: The `allowed_signers` file consists of zero or more lines of four fields separated by space. The second field is `namespaces="git"`, the third field is an OpenSSH compatible key type, and the fourth field is a base64-encoded public key.

Criterion: Every commit with parents is signed with an SSH key with the public key listed in the `allowed_signers` file of all parent commits.

Criteria for an Ungarbled Document Succession in DSGL

Criterion: The graph of Git commit records is linear.

Criterion: The initial commit is signed with an SSH key that is listed in the `allowed_signers` file within the `signed_succession` directory.

Criterion: The first field of all lines in the `allowed_signers` files is the character `*`.

Criterion: The third field of all lines in the `allowed_signers` files is the key type `ssh-ed25519`.

Criterion: All paths of all Git commit trees match the EBNF grammar for DSGL Paths (definition follows).

Criterion: Every `object` entry is only added once in the commit history.

Criterion: There are no two `object` entries whose paths correspond to edition numbers that are above or below each other (for example, `1/object` and `1/2/object` are not both present). In other words, if a Git tree directly contains an `object` entry, then it is the only direct entry in that Git tree.

DSGL Paths in Extended Backus—Naur Form (EBNF)

The following grammar is expressed in ISO/IEC 14977 Extended Backus—Naur Form (EBNF) extended further to allow an ellipsis (…) to denote a range of ASCII characters.

```
path = "signed_succession/allowed_signers" | snapshot ;
snapshot = { non_neg_int, "/" }, pos_int, "/object" ;
non_neg_int = "0" | pos_int ;
pos_int = pos_dec_digit, { dec_digit } ;
dec_digit = "0" | pos_dec_digit ;
pos_dec_digit = "1"…"9" ;
```

Discussion

Control of Document Successions

As long as authors of document successions maintain control over the allowed private SSH signing keys, a signed document succession can be moved and distributed across any Git-compatible servers, yet only the authors can extend the document succession with new editions. The capability to extend a signed document succession hinges on control of allowed private SSH signing keys, rather than control over specific accounts or servers.

Related Concepts

A *Document Succession Identifier* is an intrinsic identifier [4] [5] [6] of the initial Git commit.

Design Choice Rationales

Separation of Git History from Edition History

Representing the history of editions through means other than Git commit history is a deliberate design choice. Git commit history records all Git actions, which can lead to inflexible and complicated non-linear histories. Software Heritage automatically preserves Git commits, increasing the risk that a Git commit history could become an unintended complicated non-linear history. Non-linear Git commit histories and merge commits might be useful in certain scenarios.

Separating edition history from Git commit history also allows for future enhancements, such as retracting specific editions.

Use of Git Tree Paths Instead of Git Tags

In document successions, edition numbers are akin to software release versions, which are typically identified using Git tags. However, this specification adopts a different approach. Edition numbers are recorded with file paths in Git trees rather than Git tags. With this approach, a single latest Git commit captures a complete record of a document succession. This means copying document successions is as easy as copying Git branches. This is especially useful when consolidating records from multiple sources into a single Git repository.

In contrast, software projects, which often include release tags, are copied by cloning the entire repository. Using Git tags for edition numbers would introduce the complexity of keeping a branch and edition number tags in sync, thereby increasing the risk of problems during copying.

While branches in Git repositories are useful for managing document successions, branch names do not constitute a part of the document succession record.

Acknowledgments

Thank you to Valentin Lorentz for raising questions about design choices and pointing out an important shortcoming in how GPG digital signatures were used in the initial Git implementation of the Hidos library (version 0.3) [7].

This document has been copyedited with AI using https://copyaid.it.

History

- Copied Git storage specifics from edition 2.1 of the DSI specification.

References

1. Hidos 1.3. 2024. Available: https://archive.softwareheritage.org/swh:1:rev:0c997b13a255be2a83f150371c9364a1217fa91a;origin=https://gitlab.com/perm.pub/hidos
2. Document succession highly manual toolkit manual. 2024. Available: https://archive.softwareheritage.org/swh:1:rev:f6da04dce5f53d88c4c324c1d2546110a8d42d8a;origin=https://gitlab.com/perm.pub/dshmtm
3. Josefsson S. The Base16, Base32, and Base64 data encodings. RFC Editor; Internet Requests for Comments; RFC Editor; 2006 Oct. doi:10.17487/RFC4648
4. Heritage S. Intrinsic and extrinsic identifiers. 2020. Available: https://web.archive.org/web/20221019201056/https://www.softwareheritage.org/2020/07/09/intrinsic-vs-extrinsic-identifiers/
5. Cosmo RD, Gruenpeter M, Zacchiroli S. Referencing Source Code Artifacts: A Separate Concern in Software Citation. Computing in Science & Engineering. 2020;22: 33–43. doi: 10.1109/MCSE.2019.2963148
6. Di Cosmo R, Gruenpeter M, Zacchiroli S. Identifiers for Digital Objects: the Case of Software Source Code Preservation. iPRES 2018 - 15th International Conference on Digital Preservation. Boston, United States; 2018. pp. 1–9. Available: https://hal.archives-ouvertes.fr/hal-01865790
7. Hidos 0.3. 2022. Available: https://archive.softwareheritage.org/swh:1:rev:b963e5d2366724df6e8c34d864a7984ce4a2e1be;origin=https://gitlab.com/perm.pub/hidos