git-notary Hacker's Manual

A guided tour of the source

Chris Olstrom

March 30, 2016

Contents

1 Goals

- If we can flag the scope of changes, we can calculate a version.
- One size does not fit all. Tools should fit workflows, not the reverse.
- It should be trivial to use `git-notary` with an existing codebase.
- If `git-notary` no longer meets your needs, you should be able to drop it as easily as you picked it up.
- Versioning is metadata. Adding or changing it should not influence the commit graph. This requires that versioning be tracked out-of-band.

2 Overview

This document describes the implementation of `git-notary`.

2.1 Audience

This document is intended for the curious reader who wants to know how `git-notary` actually works. It is technical in nature, and is not required to use the program.

2.2 Objective

Consuming this document in full should impart a clear understanding of the features `git-notary` offers, how they work, and why they exist.

2.3 Background

It is assumed that you have moderate familiarity with git. For the purposes of this manual, it is enough that you know where to look for answers, and are comfortable enough with `git` to not be put off by terms like *ref* and *object*. No deep knowledge is assumed, but if you only know `clone`, `commit` and `push`, you may hit a few unfamiliar terms.

2.4 Style

Aside from the Overview, this manual makes no attempt at being formal. It is written in a conversational style, and is meant to read as a guided walkthrough of the `git-notary` source code.

3 Key Concepts

3.1 git notes

Back in 1.7.10, `git` introduced *notes* to the documentation. Due to their implementation, they were possible before then. Notes are essentially files attached to commits. They use a different ref (`git-notary` defaults to `versioning`), and each note is a file whose name is the hash of the object it references.

As an example, for a commit `abc123`, adding a note of "foo" would commit a file named "abc123" with "foo" as its contents.

3.2 functions

Where possible, functions assume reasonable defaults and accept optional arguments to override those defaults (the exception being cases where there is no obvious default). Arguments should be ordered first for consistency within `git-notary`, and second for frequency of use (as a proxy for an objective metric regarding user intuition). Where there is obvious conflict between these goals, it may be appropriate to reconsider the usage in other functions.

4 Annotated Source

4.1 Header

We'll start at the top.

4.1.1 Shebang

git-notary is implemented in pure POSIX shellscript, with only standard utilities like `awk` and `tail`. Any `bash`-isms should be considered defects.

```
#!/bin/sh
```

4.1.2 Version

`VERSION` is git-notary's internal version identifier. This is only used in the usage banner, but this should be a global value.

```
GIT_NOTARY_VERSION=2.1.1
```

4.1.3 Environment Variables

`GIT_NOTARY_NAMESPACE` is the notes ref `git-notary` should use for versioning. Since this should be consistent (unless explicitly stated otherwise) between functions, it is defined globally.

For the majority of cases, this should be invisible to users. It exists for case when the default (`versioning`) conflicts with another tool. This is expected to be rare, as the default *notes* ref is `commits`.

```
GIT_NOTARY_NAMESPACE=${GIT_NOTARY_NAMESPACE:-'versioning'}
```

4.2 notes

`notes` is the core function of `git-notary`. It produces a stream of notes and their corresponding objects. This can be consumed by other functions to do useful things.

4.2.1 usage

```
git-notary notes
```

Displays all notes on the `develop` branch since the latest tag.

```
git-notary notes feature/squash
```

Displays all notes on the `feature/squash` branch since the latest tag.

```
git-notary feature/squash develop
```

Displays all notes on the `feature/squash` branch since `develop`.

```
git-notary feature/squash develop my-namespace
```

Displays all notes on the `feature/squash` branch since `develop`, reading only notes in `my-namespace`

***2 code

```
# notes [branch] [base] [namespace]
notes() {
    BRANCH=${1-'develop'}
    BASE=${2:-$(git describe --tags --abbrev=0)}
    NAMESPACE=${3:-${GIT_NOTARY_NAMESPACE}}

    git rev-list --topo-order ${BASE}..${BRANCH} --reverse | while read OBJECT; do
        printf "${OBJECT} "
        git notes --ref=${NAMESPACE} show ${OBJECT} 2> /dev/null || echo
    done | grep -E '(MAJOR|MINOR|PATCH)$'
}
```

4.2.2 explanation

`BRANCH` is the first argument, defaulting to `develop` if not given.

`BASE` is the second argument, defaulting to the latest git tag if not given.

`NAMESPACE` is the third argument. If not given, it defaults to `$GIT_NOTARY_NAMESPACE`.

`git rev-list` produces a list of objects between two points.

`--topo-order` lists objects in *commit* order, rather than chronologically (thanks to Mark Yen for catching this in a code review).

By default, `git` lists the most recent commits first, which would be awkward for versioning. To walk *forward* through the commits, we use `--reverse`.

Each object is output to the stream, along with any notes from `$NAMESPACE` on that object.

`grep` is used to ignore any notes that do not conform to the expected format, and objects without notes.

4.3 Filters

4.3.1 squash

```
# squash
squash() {
    while read OBJECT_CHANGE; do
        OBJECT=$(echo ${OBJECT_CHANGE} | awk '{ print $1 }')
        CHANGE=$(echo ${OBJECT_CHANGE} | awk '{ print $2 }')

        case ${CHANGE} in
            MAJOR)
                RESULT=MAJOR;;
            MINOR)
                test "${RESULT}" != MAJOR && RESULT=MINOR;;
            PATCH)
                test -z "${RESULT}" && RESULT=PATCH;;
        esac
    done

    test ! -z "${RESULT}" && echo ${OBJECT} ${RESULT}
}
```

4.3.2 squeeze

```
# squeeze <up|down>
squeeze() {
    case ${1} in
        d|down|f|first|o|old*)
            DIRECTION=DOWN;;
        u|up|l|last|n|new*)
            DIRECTION=UP;;
        *)
            echo 'git-notary squeeze requires a direction (up or down)' >&2
            exit 23;;
    esac

    while read OBJECT_CHANGE; do
        OBJECT=$(echo ${OBJECT_CHANGE} | awk '{ print $1 }')
        CHANGE=$(echo ${OBJECT_CHANGE} | awk '{ print $2 }')

        if test "${CHANGE}" != "${LAST_CHANGE}"; then
            case ${DIRECTION} in
                DOWN)
                    echo ${OBJECT} ${CHANGE};;
                UP)
                    test ! -z "${LAST_OBJECT}" && echo ${LAST_OBJECT} ${LAST_CHANGE};;
            esac
        fi

        LAST_OBJECT=${OBJECT}
        LAST_CHANGE=${CHANGE}
    done

    test "${DIRECTION}" = UP && echo ${LAST_OBJECT} ${LAST_CHANGE}
}
```

4.4 Transforms

4.4.1 versions

```
# versions [initial]
versions() {
    set -o errexit
```

```
    VERSION=${1:-$(git describe --tags --abbrev=0)}

    MAJOR=$(echo ${VERSION} | awk -F . '{ print $1 }')
    MINOR=$(echo ${VERSION} | awk -F . '{ print $2 }')
    PATCH=$(echo ${VERSION} | awk -F . '{ print $3 }')

    next() {
        echo ${1} + 1 | bc
    }

    while read OBJECT_CHANGE; do
        OBJECT=$(echo ${OBJECT_CHANGE} | awk '{ print $1 }')
        CHANGE=$(echo ${OBJECT_CHANGE} | awk '{ print $2 }')

        case ${CHANGE} in
            MAJOR)
                MAJOR=$(next ${MAJOR})
                MINOR=0
                PATCH=0
                ;;
            MINOR)
                MINOR=$(next ${MINOR})
                PATCH=0
                ;;
            PATCH)
                PATCH=$(next ${PATCH})
                ;;
        esac

        VERSION=${MAJOR}.${MINOR}.${PATCH}
        echo ${OBJECT} ${VERSION}
    done
}
```

4.4.2 tags

```
# tags [--apply]
tags() {
    while read OBJECT_TAG; do
        OBJECT=$(echo ${OBJECT_TAG} | awk '{ print $1 }')
```

```
        TAG=$(echo ${OBJECT_TAG} | awk '{ print $2 }')

        if test "${1}" = '--apply'; then
            git tag ${TAG} ${OBJECT}
        else
            echo git tag ${TAG} ${OBJECT}
        fi
    done
}
```

4.5 Actions

4.5.1 fetch

```
# fetch [remote] [namespace]
fetch() {
    REMOTE=${1:-'origin'}
    NAMESPACE=${2:-${GIT_NOTARY_NAMESPACE}}

    git fetch ${REMOTE} refs/notes/${NAMESPACE}:refs/notes/${NAMESPACE}
}
```

4.5.2 push

```
# push [remote] [namespace]
push() {
    REMOTE=${1:-'origin'}
    NAMESPACE=${2:-${GIT_NOTARY_NAMESPACE}}

    git push --no-verify ${REMOTE} refs/notes/${NAMESPACE}
}
```

4.5.3 new

```
# new <major|minor|patch> [object] [namespace]
new() {
    CHANGE=$(echo ${1} | tr [:lower:] [:upper:])
    OBJECT=${2:-HEAD}
    NAMESPACE=${3:-${GIT_NOTARY_NAMESPACE}}

    if echo ${CHANGE} | grep -qE '^(MAJOR|MINOR|PATCH)$'; then
```

```
        git notes --ref=${NAMESPACE} add --message ${CHANGE} ${OBJECT}
    else
        echo MAJOR MINOR and PATCH are valid. ${CHANGE} is not.
        exit 23
    fi
}
```

4.5.4 delta

```
# delta (--squash) [object] [base] [namespace]
delta() {
    if test "${1}" = '--squash'; then
       SQUASH=true
       shift
    fi

    OBJECT=${1:-HEAD}
    BASE=${2:-$(git describe --tags --abbrev=0)}
    NAMESPACE=${3:-${GIT_NOTARY_NAMESPACE}}

    if test "${SQUASH}" = 'true'; then
        NEW=$(git-notary notes ${OBJECT} ${BASE} ${NAMESPACE} | git-notary squash | git
    else
        NEW=$(git-notary notes ${OBJECT} ${BASE} ${NAMESPACE} | git-notary versions | t
    fi

    LATEST_TAG_ON_BASE=$(git describe --tags --abbrev=0 ${BASE})
    OLD=${LATEST_TAG_ON_BASE:-'0.0.0'}
    echo "${OLD} -> ${NEW}"
}
```

4.5.5 undo

```
# undo [object] [namespace]
undo() {
    OBJECT=${1:-HEAD}
    NAMESPACE=${2:-${GIT_NOTARY_NAMESPACE}}

    git notes --ref=${NAMESPACE} remove ${OBJECT}
}
```

4.5.6 version

```
# version
version() {
    echo "git-notary ${GIT_NOTARY_VERSION}"
}
```

4.5.7 help

```
# help
help() {
    cat <<EOF
$(version)
usage:
       git-notary new <major|minor|patch> [object] [namespace]
       git-notary undo [object] [namespace]
       git-notary delta [--squash] [object] [base] [namespace]

       git-notary fetch [remote] [namespace]
       git-notary push [remote] [namespace]

       git-notary notes [branch] [base] [namespace]
       git-notary versions [initial]
       git-notary tags [--apply]
EOF
}
```

4.6 Interface

4.6.1 notary

```
# notary <command> [args]
notary() {
    COMMAND=${1}
    shift
    case ${COMMAND} in
        N|notes)
            notes ${@};;
        V|versions)
            versions ${@};;
        T|tags)
```

```
            tags ${@};;
        S|squash)
            squash ${@};;
        Z|squeeze)
            squeeze ${@};;
        n|new)
            new ${@};;
        u|undo)
            undo ${@};;
        d|delta)
            delta ${@};;
        P|push)
            push ${@};;
        f|fetch)
            fetch ${@};;
        M|major)
            new MAJOR ${@};;
        m|minor)
            new MINOR ${@};;
        p|patch)
            new PATCH ${@};;
        v|version|-v|--version)
            version;;
        h|help|-h|--help)
            help;;
        *)
            help
            exit 1
            ;;
    esac
}
```

4.7 entrypoint

```
notary ${@}
```