

# 7.4 Git Tools - Signing Your Work

## Signing Your Work

Git is cryptographically secure, but it's not foolproof. If you're taking work from others on the internet and want to verify that commits are actually from a trusted source, Git has a few ways to sign and verify work using GPG.

### GPG Introduction

First of all, if you want to sign anything you need to get GPG configured and your personal key installed.

```
$ gpg --list-keys
/Users/schacon/.gnupg/pubring.gpg
-----
pub   2048R/0A46826A 2014-06-04
uid           Scott Chacon (Git signing key) <schacon@gmail.com>
sub   2048R/874529A9 2014-06-04
```

If you don't have a key installed, you can generate one with `gpg --gen-key`.

```
$ gpg --gen-key
```

Once you have a private key to sign with, you can configure Git to use it for signing things by setting the `user.signingkey` config setting.

```
$ git config --global user.signingkey 0A46826A
```

Now Git will use your key by default to sign tags and commits if you want.

### Signing Tags

If you have a GPG private key setup, you can now use it to sign new tags. All you have to do is use `-s` instead of `-a`:

```
$ git tag -s v1.5 -m 'my signed 1.5 tag'
```

```
You need a passphrase to unlock the secret key for
user: "Ben Straub <ben@straub.cc>"
2048-bit RSA key, ID 800430EB, created 2014-05-04
```

If you run `git show` on that tag, you can see your GPG signature attached to it:

```
$ git show v1.5
tag v1.5
Tagger: Ben Straub <ben@straub.cc>
Date:   Sat May 3 20:29:41 2014 -0700

my signed 1.5 tag
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQEcBAABAgAGBQJTZbQLAAoJEF0+sviABDDrZbQH/09PFE51KPVP1anr6q1v4/Ut
LQxfojUWiLQdg2ESJItkcuweYg+kc3HCyFejeDIBw9dpXt00rY26p05qrpnG+85b
hM1/PswpPLuBSr+oCIDj5GMC2r2iEKsfv2fJbNW8iWAXVLowZRF8B0MfqX/YTMbm
ecorc4iXzQu7tupRihslbNkfvcfMnSDeSvzCpWAH17h8Wj6hhqePmLm9lAYqnKp
8S5B/1SSQuEAjRZgI4IexpZoeKGVDptPHxLLS38fozsyi0QyDyzEgJxcJQVMXxVi
RUysgqjcpT8+iQM1Pb1GfHR4XAhuOqN5Fx06PSaFZhqvWFezJ28/CLyX5q+oIVk=
=EFTF
-----END PGP SIGNATURE-----
```

```
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700
```

Change version number

## Verifying Tags

To verify a signed tag, you use `git tag -v <tag-name>`. This command uses GPG to verify the signature. You need the signer's public key in your keyring for this to work properly:

```
$ git tag -v v1.4.2.1
object 883653babd8ee7ea23e6a5c392bb739348b1eb61
type commit
tag v1.4.2.1
tagger Junio C Hamano <junkio@cox.net> 1158138501 -0700
```

GIT 1.4.2.1

```
Minor fixes since 1.4.2, including git-mv and git-http with alternates.
gpg: Signature made Wed Sep 13 02:08:25 2006 PDT using DSA key ID F3119B9A
gpg: Good signature from "Junio C Hamano <junkio@cox.net>"
gpg: aka "[jpeg image of size 1513]"
Primary key fingerprint: 3565 2A26 2040 E066 C9A7 4A7D C0C6 D9A4 F311 9B9A
```

If you don't have the signer's public key, you get something like this instead:

```
gpg: Signature made Wed Sep 13 02:08:25 2006 PDT using DSA key ID F3119B9A
gpg: Can't check signature: public key not found
error: could not verify the tag 'v1.4.2.1'
```

## Signing Commits

In more recent versions of Git (v1.7.9 and above), you can now also sign individual commits. If you're interested in signing commits directly instead of just the tags, all you need to do is add a `-s` to your `git commit` command.

```
$ git commit -a -s -m 'Signed commit'
```

```
You need a passphrase to unlock the secret key for
user: "Scott Chacon (Git signing key) <schacon@gmail.com>"
2048-bit RSA key, ID 0A46826A, created 2014-06-04
```

```
[master 5c3386c] Signed commit
 4 files changed, 4 insertions(+), 24 deletions(-)
 rewrite Rakefile (100%)
 create mode 100644 lib/git.rb
```

To see and verify these signatures, there is also a `--show-signature` option to `git log`.

```
$ git log --show-signature -1
commit 5c3386cf54bba0a33a32da706aa52bc0155503c2
gpg: Signature made Wed Jun 4 19:49:17 2014 PDT using RSA key ID 0A46826A
gpg: Good signature from "Scott Chacon (Git signing key) <schacon@gmail.com>"
Author: Scott Chacon <schacon@gmail.com>
Date: Wed Jun 4 19:49:17 2014 -0700
```

Signed commit

Additionally, you can configure `git log` to check any signatures it finds and list them in its output with the `%G?` format.

```
$ git log --pretty="format:%h %G? %aN %s"
```

```
5c3386c G Scott Chacon Signed commit
```

```
ca82a6d N Scott Chacon Change the version number
085bb3b N Scott Chacon Remove unnecessary test code
a11bef0 N Scott Chacon Initial commit
```

Here we can see that only the latest commit is signed and valid and the previous commits are not.

In Git 1.8.3 and later, `git merge` and `git pull` can be told to inspect and reject when merging a commit that does not carry a trusted GPG signature with the `--verify-signatures` command.

If you use this option when merging a branch and it contains commits that are not signed and valid, the merge will not work.

```
$ git merge --verify-signatures non-verify
fatal: Commit ab06180 does not have a GPG signature.
```

If the merge contains only valid signed commits, the merge command will show you all the signatures it has checked and then move forward with the merge.

```
$ git merge --verify-signatures signed-branch
Commit 13ad65e has a good GPG signature by Scott Chacon (Git signing key) <schacon@gmail.com>
Updating 5c3386c..13ad65e
Fast-forward
 README | 2 ++
 1 file changed, 2 insertions(+)
```

You can also use the `-S` option with the `git merge` command to sign the resulting merge commit itself. The following example both verifies that every commit in the branch to be merged is signed and furthermore signs the resulting merge commit.

```
$ git merge --verify-signatures -S signed-branch
Commit 13ad65e has a good GPG signature by Scott Chacon (Git signing key) <schacon@gmail.com>
```

```
You need a passphrase to unlock the secret key for
user: "Scott Chacon (Git signing key) <schacon@gmail.com>"
2048-bit RSA key, ID 0A46826A, created 2014-06-04
```

```
Merge made by the 'recursive' strategy.
 README | 2 ++
 1 file changed, 2 insertions(+)
```

## Everyone Must Sign

Signing tags and commits is great, but if you decide to use this in your normal workflow, you'll have to make sure that everyone on your team understands how to do so. If you don't, you'll end up spending a lot of time helping people figure out how to rewrite their commits with signed versions. Make sure you understand GPG and the benefits of signing things before adopting this as part of your standard workflow.

[prev](#) | [next](#)