email + git = <3

Git ships with built-in tools for collaborating over email. With this guide, you'll be contributing to email-driven projects like the Linux kernel, PostgreSQL, or even git itself in no time.

Step one
Installation

Let's start by installing the appropriate packages for your operating system.

Arch Linux 
Alpine Linux 
CentOS 
Debian 
Fedora 
FreeBSD 

Gentoo

Guix

macOS

All of the usual ways to install `git` on macOS include the git email tools by default, including Apple's developer tools as well as Homebrew, MacPorts, and the official git bundle from git-scm.com.

The easiest way is to just run:

```
git
```

If `git` is not installed already, you will be prompted to install the Apple command line developer tools.
It may be necessary to install missing perl SSL modules by hand:

```
sudo -H cpan Net::SMTP::SSL IO::Socket::SSL
```

Next

Nixpkgs

OpenBSD

openSUSE

pkgsrc

Solus

Ubuntu

Void Linux

Microsoft Windows

Don't see your OS here? Consult your OS documentation for installation, and once you finish this tutorial, send a patch adding yours to our git

repository.

email + git = <3

Git ships with built-in tools for collaborating over email. With this guide, you'll be contributing to email-driven projects like the Linux kernel, PostgreSQL, or even git itself in no time.

Step two
Configuration

You only need to complete this step once for each machine you intend to send emails from.

Use `git config --global --edit` to open your **global configuration file** in your default editor.

Gmail
Protonmail
Local SMTP client (msmtp, nbSMTP, sendmail)

Generic instructions

Your email provider should have instructions somewhere for **SMTP access**. Look these details up, and then add the following details from your mail provider:

```
[sendemail]
        smtpserver = mail.example.org
        smtpuser = you@example.org
        smtpencryption = tls
        smtpserverport = 587
```

Be sure to fill in the appropriate values for your email provider - you will probably only have to fill in `smtpserver` and `smtpuser`. Also, if you haven't yet, run these commands - again, making the appropriate changes:

```
git config --global user.email "you@example.org"
git config --global user.name "Your Name"
```

Next

email + git = <3

Git ships with built-in tools for collaborating over email. With this guide, you'll be contributing to email-driven projects like the Linux kernel, PostgreSQL, or even git itself in no time.

Step three
Give it a shot!

It's time to take it for a spin - we're going to send a patch. Ready?

1. **Clone the upstream repository**. No need to make a fork! We have prepared a repository for you to test with:

```
git clone https://git.sr.ht/~sircmpwn/email-test-drive
cd email-test-drive
```

2. **Make your changes**. Let's add a file with your progress so far:

```
echo "I'm about to try git send-email!" >your-name
```

Be sure to change `your-name` to your own!

3. **Commit your changes**. Check out the official (and free) Pro Git book if you don't know how to do this.

```
git add your-name
git commit -m "Demonstrate that I can use git send-email!"
```

4. **Send the patch**! If you check out the `README.md` file, you'll note that patches should be sent to ~sircmpwn/email-test-drive@lists.sr.ht.

```
git send-email --to="~sircmpwn/email-test-drive@lists.sr.ht"
HEAD^
```

   If prompted for an `In-Reply-To`, you can ignore it for now (just press enter). Follow the rest of the prompts and you've done it! If you have any problems at this step, feel free to email us for help. If it worked, you should see your email appear in the mailing list archives momentarily!

5. **Check your inbox**... someone has some feedback on your patch!

Next

Warning! Some people think that they can get away with sending patches through some means other than `git send-email`, but you can't. Your patches will be broken and a nuisance to the maintainers whose inbox they land in. Follow the golden rule: *just use git send-email*.

email + git = <3

Git ships with built-in tools for collaborating over email. With this guide, you'll be contributing to email-driven projects like the Linux kernel, PostgreSQL, or even git itself in no time.

Step four
Dealing with feedback

No one ever gets it right on the first try. Don't worry, it's all part of the process! You may receive some feedback on your patch from the maintainers of the software. This feedback will arrive in the form of a reply to your email. If you have any questions, just reply back - and remember to "reply all"! In the meantime, let's fix the patch.

In this tutorial, the feedback was generated by a bot, but feel free to reply to get the hang of it. You need to make sure your email client is configured to write emails in **plain text** before you do, and please try to avoid top posting. We also have an etiquette guide that'll help you avoid any faux pas.

1. **Make the changes**. Update the files to match the changes requested by the maintainers. We'll leave this to you.
2. **Amend your commit**. Git is designed for you to *edit* your commit history. It's not set in stone! The maintainers reviewing your work don't want to merge a patch which has mistakes, even if it's followed up by a fix, so you'll have to **amend** your previous commit:

   ```
   git commit -a --amend
   ```

   First time amending a commit? Amending and rebasing commits is an essential skill in this workflow. Check out our comprehensive guide to git rebase if you'd like to learn more.

3. **Set the default "to" address**. Let's make this easier on ourselves by setting the default email address for this repo, so we needn't enter it every time:

   ```
   git config sendemail.to "~sircmpwn/email-test-
   drive@lists.sr.ht"
   ```

4. **Send the new patch**! This time we'll use `-v2` to indicate that this the second version of this patch. If we do this again, we'll use `-v3`.

   ```
   git send-email --annotate -v2 HEAD^
   ```

   Note that we also specified the "--annotate" flag. This is going to open the email in our editor before sending it out, so we can make any changes. We're going to add some "timely commentary". Look for the "---" and add a short summary of the differences since the first patch on the next line. It should look something like this:

   ```
   Subject: [PATCH v2] Demonstrate that I can use git send-
   email!

   ---
   This fixes the issues raised from the first patch.

   src/main.c | 4 +---
   1 file changed, 1 insertion(+), 3 deletions(-)
   ```

This text gives the maintainers some extra context about your patch, but doesn't make it into the final git log. Close your editor, follow the prompts again, and that's it - you're done! Congratulations!

If you want some more tips on using git send-email, check out the next page.

Next

email + git = <3

Git ships with built-in tools for collaborating over email. With this guide, you'll be contributing to email-driven projects like the Linux kernel, PostgreSQL, or even git itself in no time.

Tips & tricks

Miscellaneous tips and tricks which may serve you well as you use git send-email.

Sending several patches at once

Use this to send the last 3 commits:

```
git send-email HEAD~3
```

Or all commits since a particular one:

```
git send-email 209210d
```

Or just the second-to-last commit:

```
git send-email -1 HEAD^^
```

See Revision Selection for more.

Specifying a sub-project

Some projects use a single mailing list for several git repositories. Try this to clarify that you're working on the "foobar" project:

```
git config format.subjectPrefix "PATCH foobar"
```

Using --annotate every time

```
git config --global sendemail.annotate yes
```

"Signing off" on your commits

Some projects, such as the Linux kernel, will ask you to "sign off" on your commits. To do this, add `--signoff` (or `-s`) to `git send-email`. To set it as the default for that git repository:

```
git config format.signOff yes
```

By doing this you acknowledge that you have read the Developer Certificate of Origin (DCO).

More approaches to authentication

This tutorial configures git in a way that causes send-email to prompt for your password, which you may find annoying. There are other ways to authenticate - the simplest of which is:

```
git config --global sendemail.smtpPass 'your password'
```

You can also have your password cached in memory for a certain period of time. To cache it for one hour, use:

```
git config --global credential.helper 'cache --timeout 3600'
```

For more sophisticated solutions, such as integration with your keyring, see the git-credential man page.

Back to the start