compiling norns

while norns is designed for easily updating to the latest software release, there are several reasons you might wish to recompile the norns programs yourself. you might want to keep on the bleeding edge by building the latest changes made to the code between official releases, either to try new features or get bug fixes as soon as possible. or you might want to make your own modifications to the software or try patches made by community members (for more details, see the extending norns page).

norns (the sound computer) comes with everything you need to edit and recompile norns (the software suite). this page gives a quick guide on how to get the latest changes and recompile. if you are interested in modifying the norns platform or knowing what's currently in the works, take a look at the github issues to see the list of open bugs and feature requests. new release builds are packaged up and released every few months or so. note that any contributions to the norns system will generally not be widely available for use in scripts until an official release comes out, since typically users update norns through the SYSTEM > UPDATE menu option.

throughout this tutorial we will be running command line programs in a shell session. the entire output printed to the terminal will be included, possibly using a `...` to skip long sections we don't need to see. in particular, the shell prompt `$` will be shown, possibly including username and server information. what this means is when the instructions say something like:

- we run:

  ```
  we@norns.local:~/norns$ ls
  build           crone.sh     matron.sh        resources    version.mk
  ...
  ```

then the only part that you actually type in is `ls`, the rest of what's shown in the docs is just the prompt that the terminal shows you when waiting for input. also, the line(s) with the prompt is the only line we type into – the other output (`build crone.sh ...`) is output from the `ls` program.

get up to date

open a terminal and connect to norns via SSH. then go to the `norns` directory using `cd norns`.

```
$ ssh we@norns.local
we@norns.local's password:
Linux norns 4.19.127-16-gb1425b1 #1 SMP PREEMPT Mon Oct 26 05:39:00 UTC 2020 armv7l

 ___ ___ ___ ___ ___
|   | . |  _|   |_ -|
|_|_|___|_| |_|_|___| monome.org/norns

we@norns.local:~$ cd norns
we@norns.local:~/norns$
```

this directory (`/home/we/norns`) contains all the source code for norns – it's a checkout of the git repository. it's also sort of the main directory for norns system files, as opposed to user files like scripts and tape recordings, which are in `/home/we/dust`. that includes the executable files for the core norns C/C++ programs including `matron` and `crone`. source code for `maiden` lives elsewhere

the git repository `/home/we/norns` is set up to point at the main norns repo on github. by default the copy of the code that's on the monome GitHub is nicknamed `origin`. to pull the latest changes on GitHub down and merge them into the local copy, we run:

```
we@norns:~/norns$ git pull origin main
From https://github.com/monome/norns
 * branch            main       -> FETCH_HEAD
Already up-to-date.
we@norns:~/norns$
```

if there are a bunch of changes up on GitHub that haven't been pulled yet, this will show you a summary of changes as well.

we also need to make sure that the submodules are all up to date. submodules are basically git repos within a git repo that help norns track dependencies on other projects, notably the main softcut library and ableton link support. run:

```
we@norns:~/norns$ git submodule update --init --recursive
Submodule path 'crone/softcut': checked out '7cce02a6f5b58a9c46bd022bd7b572e2b3218dae'
we@norns:~/norns$
```

compiling with waf

all the norns C / C++ programs are built with a build tool called `waf`. first we need to check that all dependencies are installed and set up the build environment. generally you only need to do this if a new dependency is added, which is fairly seldom. run:

```
we@norns.local:~/norns$ ./waf configure
Setting top to                           : /home/we/norns
Setting out to                           : /home/we/norns/build
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for 'g++' (C++ compiler)        : /usr/bin/g++
...
Checking for libmonome                   : yes
Checking for supercollider               : yes
Checking for program 'dpkg-architecture' : /usr/bin/dpkg-architecture
Checking boost includes                  : 1.62.0
'configure' finished successfully (2.727s)
we@norns.local:~/norns$
```

if there are missing dependencies, you may need to install them using `apt-get`. as of the current norns version with the latest image (201023 and above) all dependencies should already be installed.

assuming that you get `'configure' finished successfully` we should be already to compile:

```
we@norns.local:~/norns$ ./waf -j 1
Waf: Entering directory `/home/we/norns/build'
...
Waf: Leaving directory `/home/we/norns/build'
'build' finished successfully
```

each file that has been changed since you last compiled will be listed. the `-j 1` option instructs norns to use only one thread – using all the CPU for recompiling (the default) can make norns unresponsive.

there is no separate install step. the norns code is run from the location it's built to.

restart

assuming you get `'build' finished successfully`, you've now rebuilt norns and are ready to restart. the easiest way to do that is just to put norns to SLEEP as usual.

tl;dr

there's also a convenience script bundled with norns which automates all of the above.

via SSH/screen:

```
norns/edge.sh
systemctl restart norns*
```

via maiden:

```
os.execute("/home/we/norns/edge.sh")
;restart
```

to continue from here and learn how to make your own changes to the norns code, see extending norns.

help