surplus on wheels: WhatsApp Bridge

WhatsApp Bridge for surplus on wheels (s+ow)

s+ow bridges are defined in a file named `$HOME/.s+ow-bridges`. each command in the file is run, and comma-seperated target chat IDs are passed using stdin.

this bridge recognises targets prefixed with `wa:`.

```
wa:<chat id>,...
```

installation

from a pre-built binary

```
wget -O- https://surplus.joshwel.co/whatsapp.sh | sh
```

> **Note**
>
> if `wget` throws a 404, see backup links

building from source

on Termux

1. clone the repository at either `https://forge.joshwel.co/mark/surplus` or `https://github.com/markjoshwel/surplus`, and navigate to `src/spow-whatsapp-bridge` within the cloned repository

   ```
   git clone https://forge.joshwel.co/mark/surplus
   cd surplus/src/spow-whatsapp-bridge
   ```

2. build the bridge:

   ```
   go build
   ```

   for compatibility with the documentations' instructions as-is, rename the built binary to `s+ow-whatsapp-bridge`

   ```
   mv spow-whatsapp-bridge s+ow-whatsapp-bridge
   ```

3. send the built binary over to your Termux environment, and then move it into the `$HOME/.local/bin/` folder. if it doesn't exist, make it with `mkdir` and ensure that the folder is in your `PATH` variable either using your `.profile`, `.bashrc` or whatever file is sourced when opening your shell

anywhere else

for usage on Termux, see if the Android NDK supports your platform

1. grab a copy of the NDK, and extract it somewhere. navigate to `<ndk folder>/toolchains/llvm/prebuilt/<your platform>/bin` and look for a suitable `clang` executable, as it will be your CGO compiler

   ```
   m@csp:~/android-ndk-r26d/toolchains/llvm/prebuilt/linux-x86_64/bin$ ls *clang
   aarch64-linux-android21-clang  aarch64-linux-android30-clang     ...
   aarch64-linux-android22-clang  aarch64-linux-android31-clang
   aarch64-linux-android23-clang  aarch64-linux-android32-clang
   aarch64-linux-android24-clang  aarch64-linux-android33-clang
   aarch64-linux-android25-clang  aarch64-linux-android34-clang
   aarch64-linux-android26-clang  armv7a-linux-androideabi21-clang
   aarch64-linux-android27-clang  armv7a-linux-androideabi22-clang
   aarch64-linux-android28-clang  armv7a-linux-androideabi23-clang
   aarch64-linux-android29-clang  armv7a-linux-androideabi24-clang
   ```

   the example output is not exhaustive and is cut short for brevity and example, do take a look at your downloaded NDK archive for what executables are available to you

   many executables are present, so choose a) what architecture you will build for (more often than not it's `aarch64`), and b) what target android api are you building for

   if you're building for yourself, pick an api level/version that correlates to your devices' android version. as an example, my device runs on an ARM processor (`aarch64`) and runs Android 14, which is api level 34. (`android34`) as such, i would use the `aarch64-linux-android34-clang` binary

2. clone the repository at either `https://forge.joshwel.co/mark/surplus` or `https://github.com/markjoshwel/surplus`, and navigate to `src/spow-whatsapp-bridge` within the cloned repository

   ```
   git clone https://forge.joshwel.co/mark/surplus
   cd surplus/src/spow-whatsapp-bridge
   ```

3. build the bridge:

   ```
   CC="&lt;path to android ndk clang executable&gt;" GOOS=android GOARCH=arm64 CGO_ENABLED=1 go build
   ```

   for compatibility with the documentations' instructions as-is, rename the built binary to `s+ow-whatsapp-bridge`

   ```
   mv spow-whatsapp-bridge s+ow-whatsapp-bridge
   ```

4. send the built binary over to your Termux environment, and then move it into the `$HOME/.local/bin/` folder. if it doesn't exist, make it with `mkdir` and ensure that the folder is in your `PATH` variable either using your `.profile`, `.bashrc` or whatever file is sourced when opening your shell

post-installation setup

1. log into WhatsApp:

   ```
   s+ow-whatsapp-bridge login
   ```

   give it a minute or two to sync your history. once the screen stops scrolling, you can safely exit with Ctrl+D or Ctrl+C.

2. find out what chats you want the bridge to target:

   ```
   s+ow-whatsapp-bridge list
   ```

   > **Note**
   >
   > for sending to individuals: their IDs are their internationalised phone numbers ending in `@s.whatsapp.net`
   >
   > example: `+65 9123 4567` is `6591234567@s.whatsapp.net`

   then, note these down, prefixed with `wa:`, to them to your `SPOW_TARGETS` variable in your s+ow cron job

3. finally, add the following to your $HOME/.s+ow-bridges file:

   ```
   s+ow-whatsapp-bridge
   ```

updating

to keep updated as whatsmeow, the library the bridge depends on, has to keep updated with the WhatsApp web multidevice API, you can either:

1. rebuild when a weekly release comes out,
2. or rely on the weekly continuous deployment builds

to use the weekly builds without building from scratch every time,

> **Note**
>
> this will pull the latest binary, around 20 megabytes in size, every day. if your network or data plan may not take kindly to this, feel free to adjust the cron entry as you wish, or to one that runs once a week instead:
>
> ```
> 0 0 * * 0      bash -l -c "s+ow-whatsapp-bridge-update"
> ```

usage

- `s+ow-whatsapp-bridge`
  normal usage; sends latest message to wa:-prefixed targets given in stdin
- `s+ow-whatsapp-bridge login`
  logs in to WhatsApp
- `s+ow-whatsapp-bridge pair-phone`
  logs in to WhatsApp using a phone number
- `s+ow-whatsapp-bridge reconnect`
  reconnects the client
- `s+ow-whatsapp-bridge logout`
  logs out of WhatsApp
- `s+ow-whatsapp-bridge list`
  lists all group chats and their IDs.

  for sending to individuals: their IDs are their internationalised phone numbers ending in `@s.whatsapp.net`

  example: `+65 9123 4567` is `6591234567@s.whatsapp.net`

verifying a pre-built binary

> **Note**
>
> if you installed the bridge through an installation script, it would have already
>
> and if the script or `s+ow-whatsapp-bridge-update` throws an error about failing verification, you can use the environment variable ``

TODO

versioning scheme

from `v2.2024.25`, the bridge is now versioned with a modified calendar versioning scheme of `MAJOR.YEAR.ISOWEEK`, where the `MAJOR` version segment will be bumped with codebase changes, whereas the `YEAR` and `ISOWEEK` segments will represent the time of which the release was built at

licence

the s+ow Telegram Bridge is free and unencumbered software released into the public domain. for more information, see licences.