Release Process

Branch updates

Before every release candidate

- Update translations see translation_process.md.
- Update release candidate version in `configure.ac` (`CLIENT_VERSION_RC`).
- Update manpages (after rebuilding the binaries), see gen-manpages.py.

Before every major and minor release

- Update bips.md to account for changes since the last release (don't forget to bump the version number on the first line).
- Update version in `configure.ac` (don't forget to set `CLIENT_VERSION_RC` to `0`).
- Update manpages (see previous section)
- Write release notes (see "Write the release notes" below).

Before every major release

- On both the master branch and the new release branch:
  - update `CLIENT_VERSION_MAJOR` in `configure.ac`
- On the new release branch in `configure.ac` (see this commit):
  - set `CLIENT_VERSION_MINOR` to `0`
  - set `CLIENT_VERSION_BUILD` to `0`
  - set `CLIENT_VERSION_IS_RELEASE` to `true`

Before branch-off

- Update hardcoded seeds, see this pull request for an example.
- Update `src/chainparams.cpp` m_assumed_blockchain_size and m_assumed_chain_state_size with the current size plus some overhead (see this for information on how to calculate them).
- Update `src/chainparams.cpp` chainTxData with statistics about the transaction count and rate. Use the output of the `getchaintxstats` RPC, see this pull request for an example. Reviewers can verify the results by running `getchaintxstats <window_block_count> <window_final_block_hash>` with the `window_block_count` and `window_final_block_hash` from your output.
- Update `src/chainparams.cpp` nMinimumChainWork and defaultAssumeValid (and the block height comment) with information from the `getblockheader` (and `getblockhash`) RPCs.
  - The selected value must not be orphaned so it may be useful to set the value two blocks back from the tip.
  - Testnet should be set some tens of thousands back from the tip due to reorgs there.
  - This update should be reviewed with a reindex-chainstate with assumevalid=0 to catch any defect that causes rejection of blocks in the past history.
- Clear the release notes and move them to the wiki (see "Write the release notes" below).
- Translations on Transifex
  - Create a new resource named after the major version with the slug `[bitcoin.qt-translation-<RRR>x]` , where `RRR` is the major branch number padded with zeros. Use `src/qt/locale/bitcoin_en.xlf` to create it.
  - In the project workflow settings, ensure that Translation Memory Fill-up is enabled and that Translation Memory Context Matching is disabled.
  - Update the Transifex slug in `.tx/config` to the slug of the resource created in the first step. This identifies which resource the translations will be synchronized from.
  - Make an announcement that translators can start translating for the new version. You can use one of the previous announcements as a template.
  - Change the auto-update URL for the resource to `master` , e.g. `https://raw.githubusercontent.com/bitcoin/bitcoin/master/src/qt/locale/bitcoin_en.xlf` . (Do this only after the previous steps, to prevent an auto-update from interfering.)

After branch-off (on the major release branch)

- Update the versions.
- Create the draft, named "*version* Release Notes Draft", as a collaborative wiki.
- Clear the release notes: `cp doc/release-notes-empty-template.md doc/release-notes.md`
- Create a pinned meta-issue for testing the release candidate (see this issue for an example) and provide a link to it in the release announcements where useful.
- Translations on Transifex
  - Change the auto-update URL for the new major version's resource away from `master` and to the branch, e.g. `https://raw.githubusercontent.com/bitcoin/bitcoin/<branch>/src/qt/locale/bitcoin_en.xlf`. Do not forget this or it will keep tracking the translations on master instead, drifting away from the specific major release.

Before final release

- Merge the release notes from the wiki into the branch.
- Ensure the "Needs release note" label is removed from all relevant pull requests and issues.

Tagging a release (candidate)

To tag the version (or release candidate) in git, use the `make-tag.py` script from bitcoin-maintainer-tools. From the root of the repository run:

```
../bitcoin-maintainer-tools/make-tag.py v(new version, e.g. 23.0)
```

This will perform a few last-minute consistency checks in the build system files, and if they pass, create a signed tag.

Building

First time / New builders

Install Guix using one of the installation methods detailed in contrib/guix/INSTALL.md.

Check out the source code in the following directory hierarchy.

```
cd /path/to/your/toplevel/build
git clone https://github.com/bitcoin-core/guix.sigs.git
git clone https://github.com/bitcoin-core/bitcoin-detached-sigs.git
git clone https://github.com/bitcoin/bitcoin.git
```

Write the release notes

Open a draft of the release notes for collaborative editing at https://github.com/bitcoin-core/bitcoin-devwiki/wiki.

For the period during which the notes are being edited on the wiki, the version on the branch should be wiped and replaced with a link to the wiki which should be used for all announcements until `-final`.

Generate the change log. As this is a huge amount of work to do manually, there is the `list-pulls` script to do a pre-sorting step based on github PR metadata. See the documentation in the README.md.

Generate list of authors:

```
git log --format='- %aN' v(current version, e.g. 0.20.0)..v(new version, e.g. 0.20.1) | sort -fiu
```

Setup and perform Guix builds

Checkout the Bitcoin Core version you'd like to build:

```
pushd ./bitcoin
SIGNER='(your builder key, ie bluematt, sipa, etc)'
VERSION='(new version without v-prefix, e.g. 0.20.0)'
git fetch origin "v${VERSION}"
git checkout "v${VERSION}"
popd
```

Ensure your guix.sigs are up-to-date if you wish to `guix-verify` your builds against other `guix-attest` signatures.

```
git -C ./guix.sigs pull
```

Create the macOS SDK tarball (first time, or when SDK version changes)

Create the macOS SDK tarball, see the macdeploy instructions for details.

Build and attest to build outputs

Follow the relevant Guix README.md sections:

- Building
- Attesting to build outputs

Verify other builders' signatures to your own (optional)

- Add other builders keys to your gpg keyring, and/or refresh keys
- Verifying build output attestations

Commit your non codesigned signature to guix.sigs

```
pushd ./guix.sigs
git add "${VERSION}/${SIGNER}"/noncodesigned.SHA256SUMS{,.asc}
git commit -m "Add attestations by ${SIGNER} for ${VERSION} non-codesigned"
git push  # Assuming you can push to the guix.sigs tree
popd
```

Codesigning

macOS codesigner only: Create detached macOS signatures (assuming signapple is installed and up to date with master branch)

```
tar xf bitcoin-osx-unsigned.tar.gz
./detached-sig-create.sh /path/to/codesign.p12
Enter the keychain password and authorize the signature
signature-osx.tar.gz will be created
```

Windows codesigner only: Create detached Windows signatures

```
tar xf bitcoin-win-unsigned.tar.gz
./detached-sig-create.sh -key /path/to/codesign.key
Enter the passphrase for the key when prompted
signature-win.tar.gz will be created
```

Windows and macOS codesigners only: test code signatures

It is advised to test that the code signature attaches properly prior to tagging by performing the `guix-codesign` step. However if this is done, once the release has been tagged in the bitcoin-detached-sigs repo, the `guix-codesign` step must be performed again in order for the guix attestation to be valid when compared against the attestations of non-codesigner builds.

Windows and macOS codesigners only: Commit the detached codesign payloads

```
pushd ./bitcoin-detached-sigs
# checkout the appropriate branch for this release series
rm -rf ./*
tar xf signature-osx.tar.gz
tar xf signature-win.tar.gz
git add -A
git commit -m "point to ${VERSION}"
git tag -s "v${VERSION}" HEAD
git push the current branch and new tag
popd
```

Non-codesigners: wait for Windows and macOS detached signatures

- Once the Windows and macOS builds each have 3 matching signatures, they will be signed with their respective release keys.
- Detached signatures will then be committed to the bitcoin-detached-sigs repository, which can be combined with the unsigned apps to create signed binaries.

Create the codesigned build outputs

- Codesigning build outputs

Verify other builders' signatures to your own (optional)

- Add other builders keys to your gpg keyring, and/or refresh keys
- Verifying build output attestations

Commit your codesigned signature to guix.sigs (for the signed macOS/Windows binaries)

```
pushd ./guix.sigs
git add "${VERSION}/${SIGNER}"/all.SHA256SUMS{,.asc}
git commit -m "Add attestations by ${SIGNER} for ${VERSION} codesigned"
git push  # Assuming you can push to the guix.sigs tree
popd
```

After 3 or more people have guix-built and their results match

Combine the `all.SHA256SUMS.asc` file from all signers into `SHA256SUMS.asc`:

```
cat "$VERSION"/*/all.SHA256SUMS.asc > SHA256SUMS.asc
```

- Upload to the bitcoincore.org server (`/var/www/bin/bitcoin-core-${VERSION}/`):

    1. The contents of each `./bitcoin/guix-build-${VERSION}/output/${HOST}/` directory, except for `*-debug*` files.

       Guix will output all of the results into host subdirectories, but the SHA256SUMS file does not include these subdirectories. In order for downloads via torrent to verify without directory structure modification, all of the uploaded files need to be in the same directory as the SHA256SUMS file.

The `*-debug*` files generated by the guix build contain debug symbols for troubleshooting by developers. It is assumed that anyone that is interested in debugging can run guix to generate the files for themselves. To avoid end-user confusion about which file to pick, as well as save storage space *do not upload these to the bitcoincore.org server, nor put them in the torrent.*

```
find guix-build-${VERSION}/output/ -maxdepth 2 -type f -not -name
"SHA256SUMS.part" -and -not -name "*debug*" -exec scp {}
user@bitcoincore.org:/var/www/bin/bitcoin-core-${VERSION} \;
```

2. The `SHA256SUMS` file
3. The `SHA256SUMS.asc` combined signature file you just created

- Create a torrent of the `/var/www/bin/bitcoin-core-${VERSION}` directory such that at the top level there is only one file: the `bitcoin-core-${VERSION}` directory containing everything else. Name the torrent `bitcoin-${VERSION}.torrent` (note that there is no `-core-` in this name).

  Optionally help seed this torrent. To get the `magnet:` URI use:

  ```
  transmission-show -m <torrent file>
  ```

  Insert the magnet URI into the announcement sent to mailing lists. This permits people without access to `bitcoincore.org` to download the binary distribution. Also put it into the `optional_magnetlink:` slot in the YAML file for bitcoincore.org.

- Update other repositories and websites for new version
  - bitcoincore.org blog post
  - bitcoincore.org maintained versions update: table
  - Delete post-EOL release branches and create a tag `v${branch_name}-final`.
  - Delete "Needs backport" labels for non-existing branches.
  - bitcoincore.org RPC documentation update
    - Install golang
    - Install the new Bitcoin Core release
    - Run bitcoind on regtest
    - Clone the bitcoincore.org repository
    - Run: `go run generate.go` while being in `contrib/doc-gen` folder, and with bitcoin-cli in PATH
    - Add the generated files to git
  - Update packaging repo
    - Push the flatpak to flathub, e.g. https://github.com/flathub/org.bitcoincore.bitcoin-qt/pull/2
    - Push the snap, see https://github.com/bitcoin-core/packaging/blob/master/snap/build.md
  - This repo
    - Archive the release notes for the new version to `doc/release-notes/` (branch `master` and branch of the release)

        - Create a new GitHub release with a link to the archived release notes

- Announce the release:
    - bitcoin-dev and bitcoin-core-dev mailing list
    - Bitcoin Core announcements list https://bitcoincore.org/en/list/announcements/join/
    - Bitcoin Core Twitter https://twitter.com/bitcoincoreorg
    - Celebrate

Additional information

How to calculate `m_assumed_blockchain_size` and `m_assumed_chain_state_size`

Both variables are used as a guideline for how much space the user needs on their drive in total, not just strictly for the blockchain. Note that all values should be taken from a **fully synced** node and have an overhead of 5-10% added on top of its base value.

To calculate `m_assumed_blockchain_size`:

- For `mainnet` -> Take the size of the data directory, excluding `/regtest` and `/testnet3` directories.
- For `testnet` -> Take the size of the `/testnet3` directory.

To calculate `m_assumed_chain_state_size`:

- For `mainnet` -> Take the size of the `/chainstate` directory.
- For `testnet` -> Take the size of the `/testnet3/chainstate` directory.

Notes:

- When taking the size for `m_assumed_blockchain_size`, there's no need to exclude the `/chainstate` directory since it's a guideline value and an overhead will be added anyway.
- The expected overhead for growth may change over time, so it may not be the same value as last release; pay attention to that when changing the variables.