If you're a TS maintainer, you can respond to a PR with a comment similar to

```
@typescript-bot test this
```

to trigger a specialized on-demand build on the PR.

The currently recognized commands are:

- `test this` - This runs the internal RWC suite against the PR (this takes around 20 minutes). If the tests fail, a diff will be produced and a PR against the internal RWC suite repo will be made with the diff.
- `run dt` - The runs the definitely typed linter using the PR TS build sharded across 4 worker containers (this takes around 25 minutes).
- `run dt slower` - This is the same as the above, but only on a single worker (this takes around 90 minutes). This is useful if the results aren't needed promptly and the build queue is busy.
- `user test this` - This runs the nightly-tested `user` suite against the PR (this takes around 30 minutes). If this fails, the bot will attempt to open a PR against the triggering PR with the baseline diffs - this will only succeed if the PR is for a branch on `Microsoft/TypeScript`, or if the fork has PRs enabled (so, bit of advice if you plan to use this on your branches: enable PRs on your personal fork).
- `user test this slower` - The same as the above, but run on only a single container, meaning it takes around 1h 30m. The nightly run is run using this build.
- `user test this inline` - Similar to `user test this`, but with a second run for comparing to master, and the bot posts results on the issue. This is newer than `user test this` and is now the preferred option.
- `perf test` - This queues a build on our perf server using the code from the PR - once started (which will only happen once any currently running build is done), this takes around 40 minutes. The bot should post the results of the perf test run back into the triggering PR once done.
- `perf test faster` - This is the same as the above, but only runs the node 12 host tests - so the results should be less complete, but come back in around 1/3rd the time.
- `pack this` - This creates a build which does a build, runs an LKG, runs normal tests, and then packs the result into an installable tarball (which can be downloaded from the build artifacts on the azure pipelines build status page), perfect for installing with `npm i <URL to tarball>` to test with.
- `cherry-pick this to branchname` - This launches a task to squash the commits from the PR and then open a new PR that cherry-picks the change into branch `branchname`. This takes about 5 minutes as the build agent needs to clone the input PR. The bot should reply if something goes wrong, or otherwise once the new PR is open.
- `cherry-pick this to branchname and LKG` - Same as above, but an LKG commit will be added onto the PR after the squashed cherry-pick commit.
- `run repros` - Triggers inline code repro workflow

In addition, there are a small suite of commands which work in *any* comment and relate to release management. You can see how these are typically used in our documented comment command sequence:

- `create release-X.Y` This makes a `release-X.Y` branch (replace `X.Y` with your desired version) with the `package.json` version set to `X.Y.0-beta`, the `corePublic` `versionMajorMinor` set to `X.Y`, and the full `ts.version` string set to `X.Y.0-beta`, and updates the accompanying baselines. An LKG is then performed. This new branch is directly pushed to `microsoft/TypeScript`. In short, this fully sets up a new release branch to be ready to publish a beta.
- `bump release-X.Y` This bumps the version (`0-beta` -> `1-rc` -> `2` -> `3` and so on) on the specified branch and captures a new LKG, essentially preparing the branch for a new release.

- `sync release-X.Y` This merges `master` into the specified branch; this is useful for syncing the branch with `master` in the period between the beta and rc.

A single comment may contain multiple commands, so long as each is prefixed with a call to `@typescript-bot`.

The source of the webhook running the bot is currently available here.