VS Code Smoke Test

Make sure you are on **Node v12.x**.

Quick Overview

```
# Build extensions in the VS Code repo (if needed)
yarn && yarn compile

# Dev (Electron)
yarn smoketest

# Dev (Web - Must be run on distro)
yarn smoketest --web --browser [chromium|webkit]

# Build (Electron)
yarn smoketest --build <path to latest version>
example: yarn smoketest --build /Applications/Visual\ Studio\ Code\ -\ Insiders.app

# Build (Web - read instructions below)
yarn smoketest --build <path to server web build (ends in -web)> --web --browser
[chromium|webkit]

# Remote (Electron)
yarn smoketest --build <path to latest version> --remote
```

* This step is necessary only when running without `--build` and OSS doesn't already exist in the `.build/electron` directory.

Running for a release (Endgame)

You must always run the smoketest version that matches the release you are testing. So, if you want to run the smoketest for a release build (e.g. `release/1.22`), you need to check out that version of the smoke tests too:

```
git fetch
git checkout release/1.22
yarn && yarn compile
yarn --cwd test/smoke
```

Web

There is no support for testing an old version to a new one yet. Instead, simply configure the `--build` command line argument to point to the absolute path of the extracted server web build folder (e.g. `<rest of path here>/vscode-server-darwin-x64-web` for macOS). The server web build is available from the builds page (see previous subsection).

macOS: if you have downloaded the server with web bits, make sure to run the following command before unzipping it to avoid security issues on startup:

```
xattr -d com.apple.quarantine <path to server with web folder zip>
```

Note: make sure to point to the server that includes the client bits!

Debug

- `--verbose` logs all the low level driver calls made to Code;
- `-f PATTERN` (alias `-g PATTERN`) filters the tests to be run. You can also use pretty much any mocha argument;
- `--headless` will run playwright in headless mode when `--web` is used.

Note: you can enable verbose logging of playwright library by setting a `DEBUG` environment variable before running the tests (https://playwright.dev/docs/debug#verbose-api-logs), for example to `pw:browser`.

Develop

```
cd test/smoke
yarn watch
```

Troubleshooting

Error: Could not get a unique tmp filename, max tries reached

On Windows, check for the folder `C:\Users\<username>\AppData\Local\Temp\t`. If this folder exists, the `tmp` module can't run properly, resulting in the error above. In this case, delete the `t` folder.

Pitfalls

- Beware of workbench **state**. The tests within a single suite will share the same state.
- Beware of **singletons**. This evil can, and will, manifest itself under the form of FS paths, TCP ports, IPC handles. Whenever writing a test, or setting up more smoke test architecture, make sure it can run simultaneously with any other tests and even itself. All test suites should be able to run many times in parallel.
- Beware of **focus**. **Never** depend on DOM elements having focus using `.focused` classes or `:focus` pseudo-classes, since they will lose that state as soon as another window appears on top of the running VS Code window. A safe approach which avoids this problem is to use the `waitForActiveElement` API. Many tests use this whenever they need to wait for a specific element to *have focus*.
- Beware of **timing**. You need to read from or write to the DOM... but is it the right time to do that? Can you 100% guarantee that `input` box will be visible at that point in time? Or are you just hoping that it will be so? Hope is your worst enemy in UI tests. Example: just because you triggered Quick Access with `F1`, it doesn't mean that it's open and you can just start typing; you must first wait for the input element to be in the DOM as well as be the current active element.
- Beware of **waiting**. **Never** wait longer than a couple of seconds for anything, unless it's justified. Think of it as a human using Code. Would a human take 10 minutes to run through the Search viewlet smoke test? Then, the computer should even be faster. **Don't** use `setTimeout` just because. Think about what you should wait for in the DOM to be ready and wait for that instead.