Click to verify

GNU Core Utilities for Building on Windows To build software on Windows, you'll likely need GNU make and several other GNU Core Utilities like touch, rm, cp, sed, test, tee, echo, and more. Some of these utilities require Bash features to function correctly. Many users turn to the POSIX emulators, such as Cygwin or GnuWin (now abandoned), to provide a Unix-like environment on Windows. While both offer compatibility with POSIX standards, their performance varies. ezwinports is a favorite among developers due to its speed and reliability. Although it does not include Bash, you can use Make from ezwinports alongside Bash from Cygwin or MSYS2. MSYS 1.19 has been discontinued but was once bundled with MinGW packages that included an outdated version of make.exe. Use the more recent mingw32-make.exe instead, which might require adjustments to your environment setup. When using makefiles, be aware of recursive expanded variables versus lazy initialization. If not handled correctly, this can lead to infinite loops in your build process. For instance, consider the following lines: `VAR1 = 10` and `VAR1 = $(VAR1) + 100`. Make will initially assign a value from the right-hand side to the left-hand side variable without expanding it. However, when make reads back `$(VAR1)` which has been assigned on the left-hand side, this process can repeat indefinitely. In contrast, expanded variables are instantiated immediately upon reading. In the provided example, `VAR1 = $(VAR1) + 100` would result in an output of `10 + 100`, as expected. When trying to build software with makefiles in Windows environments like Cygwin or MSYS2, ensure that your system is correctly configured and that you understand how these environments handle variables and file paths. Some systems, such as those using Bash, require careful setup to work seamlessly across different platforms. For instance, consider the scenario of running a makefile from within Windows 10, which results in the error `'make' is not recognized as an internal or external command, operable program or batch file`. This issue may stem from an incorrect environment path variable setting or a missing executable. To resolve this, add the correct path to your system's `PATH` variable for the make executable. Moreover, be cautious of browsers and their requirements for self-signed certificates in order to ensure that they are compatible with various clients and systems. To create a self-signed certificate with OpenSSL, follow these steps: Steps 1 and 5 allow you to avoid the third-party authority, acting as your own authority. However, some browsers don't trust self-signed certificates from Android's default browser, making it unsuitable for platforms like this. The W3C's WebAppSec Working Group is addressing the issue of browsers not trusting self-signed certificates in IoT devices. For example, when connecting to a thermostat or refrigerator, the outcome will be poor user experience. To create a self-signed certificate using OpenSSL: 1. Create a configuration file with OpenSSL using the `example-com.conf` file. 2. Use the `-config` option and generate a self-signed certificate with the `-new`, `-x509`, `-sha256`, `-newkey rsa:2048`, `-nodes`, `-days 365`, and `-out example-com.cert.pem` options. 3. Generate a signing request using OpenSSL with the `example-com.conf` file, `-new`, `-sha256`, `-newkey rsa:2048`, `-nodes`, `-keyout example-com.key.pem`, `-days 365`, and `-out example-com.req.pem` options. The configuration file has sections for both certificate generation (`req`) and signing request creation (`req_ext`). The `subjectAltName` is used to specify DNS names in the SAN instead of the Common Name (CN). The package manager uv, released in 2024, offers a convenient solution. It allows you to use any Python version shown by the command uv python list, which it will install automatically, if required, separately from your system Python installation (in Linux under ~/.local/share/uv). To create an empty virtual environment using a specific Python version, such as 3.8, the simplest command is: uv venv --python 3.8 It's worth noting that this command creates an environment without pip and other packages included in the standard library python -m venv. If you want to use uv only for choosing the Python version, but not for package management, you can run the standard venv command with a specific Python version: uv run --python 3.8 python -m venv Alternatively, if you plan to use another Python version outside of the virtual environment, you can use uv python install 3.8 to create a symbolic link in your PATH (in Linux under ~/.local/bin), allowing you to then use python3.8 and run python -m venv.