Explicit Linker Path Display - Improvements

The Issue Identified

The other AI correctly pointed out that the profile should **always show the actual linker path explicitly**, not rely on environment variables that might be undefined or unclear.

The Problem

Some functions were:

- Not showing the linker at all (MSYS2 functions)
- Showing generic names like "MSVC link.exe" without full path

This made it unclear:

- What linker is actually being used?
- Where is the linker located?
- Is the environment correctly configured?

The Fix Applied

1. MSYS2 GCC Function

Before:

```
Write-Host "  Compiler: $msys64Root\ucrt64\bin\gcc.exe" -ForegroundColor Green
# No linker shown!
```

After:

```
Write-Host "  Compiler: $msys64Root\ucrt64\bin\gcc.exe" -ForegroundColor Green
Write-Host "  Linker: GNU ld (auto-detected by GCC)" -ForegroundColor Green
```

2. MSYS2 Clang Function

Before:

```
Write-Host "  Compiler: $msys64Root\ucrt64\bin\clang.exe" -ForegroundColor Green
# No linker shown!
```

After:

```
Write-Host "  Compiler: $msys64Root\ucrt64\bin\clang.exe" -ForegroundColor Green
Write-Host "  Linker: LLVM lld (auto-detected by Clang)" -ForegroundColor Green
```

3. Default Environment Display

Before:

```
Write-Host "  Compiler:  MSVC cl.exe" -ForegroundColor Green
Write-Host "  Linker:    MSVC link.exe" -ForegroundColor Green
```

After:

```
Write-Host "  Compiler:  $msvcBinPath\cl.exe" -ForegroundColor Green
Write-Host "  Linker:    $msvcBinPath\link.exe" -ForegroundColor Green
```

4. Environment Switching Help

Before:

```
use-gcc          - MSYS2 GCC (cmake-msys + ninja-msys + gcc.exe)
use-clang-msys   - MSYS2 Clang (cmake-msys + ninja-msys + clang.exe)
```

After:

```
use-gcc          - MSYS2 GCC (gcc.exe + GNU ld)
use-clang-msys   - MSYS2 Clang (clang.exe + LLVM lld)
```

Philosophy: Always Show The Truth

Following your original profile's approach:

- **Always show actual paths** when possible
- **Always mention the linker** for every environment
- **Be explicit** about what's being used
- **No guessing** - show the truth directly

Example Output Now

When you run `use-clang-win`:

```
Switching to Windows Clang-cl (MSVC backend) build environment...
  CMake: D:\Programs\cmake\bin\cmake.exe
  Ninja: D:\Programs\cmake\bin\ninja.exe
  Compiler: D:\Programs\clang\bin\clang-cl.exe (MSVC-compatible)
  Linker: D:\Programs\clang\bin\lld-link.exe
```

When you run `use-gcc`:

```
Switching to MSYS2/GCC build environment...
  CMake: D:\Programs\msys64\ucrt64\bin\cmake.exe
  Ninja: D:\Programs\msys64\ucrt64\bin\ninja.exe
  Compiler: D:\Programs\msys64\ucrt64\bin\gcc.exe
  Linker: GNU ld (auto-detected by GCC)
```

When profile loads (default environment):

```
Default Build Environment (for Cargo):
  CMake:     Windows CMake 4.2.3
  Generator: Ninja (Windows)
  Compiler:  D:\dev\msvc\VC\Tools\MSVC\14.50.35717\bin\Hostx64\x64\cl.exe
  Linker:    D:\dev\msvc\VC\Tools\MSVC\14.50.35717\bin\Hostx64\x64\link.exe
```

Why This Matters

When debugging build issues, you need to know **exactly** what tools are being used:

- Is it using the right linker?
- Where is it located?
- Is PATH correct?

Are environment variables set right?

Without explicit paths, you're **guessing**. With explicit paths, you **know**.

Note on Auto-Detected Linkers

For GCC and MSYS2 Clang, the linker is **auto-detected** by the compiler:

- GCC automatically uses `ld` (GNU linker)
- Clang automatically uses `lld` (LLVM linker)

We don't set `CMAKE_LINKER` for these because:

1. The compiler knows which linker to use
2. Setting it explicitly can sometimes cause issues
3. It's more reliable to let the compiler decide

But we **do show** which linker will be used so you're informed!

Summary of Improvements

Function	Before	After
`use-msvc`	Shows linker	Shows full path
`use-clang-win`	Shows linker	Shows full path
`use-gcc`	No linker shown	Shows linker (auto-detected)
`use-clang-msys`	No linker shown	Shows linker (auto-detected)
Default environment	~ Generic name	Shows full path

All environment switching functions now provide **complete, explicit information** about the toolchain being used.