flex Your ENV with FLOX

```
#include <stdio.h>

int main()
{
  int fahr;

  for (fahr = 300; fahr >= 0; fahr = fahr - 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
```

```
#!/usr/bin/env bash
set -euo pipefail

echo "============================="
echo "🟤 FreeBSD QEMU Setup Script"
echo "============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/releases/ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")

  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using $manager..."
      ${PKG_MANAGERS[$manager]} "${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "📌 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): " CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): " USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] && CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): " USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] && RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default: $DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] && DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi
  fi

  if [ -e /dev/kvm ]; then
    echo "✅ KVM acceleration available."
  else
    echo "❌ KVM still not available. Enable VT-x/AMD-V in BIOS/UEFI."
    exit 1
  fi
}

# ---------- MAIN ----------
customize_resources
check_dependencies
check_kvm

echo "✅ Environment ready. Continue with ISO..."
```

Agenta

1 . Introduction Or it’s works on my machine

2. The Problem with Global Package Managers

apt , snap , pacman

3. Introducing **Flox**

4. Explore the **core capabilities** that set FLOX apart

5. Let’s mess with ENV(WORKSHOP)

```
#include <stdio.h>

int main()
{
  int fahr;

  for (fahr = 300; fahr
>= 0; fahr = fahr - 20)
        printf("%3d
%6.1f\n",       fahr,
(5.0/9.0)*(fahr-32));
```

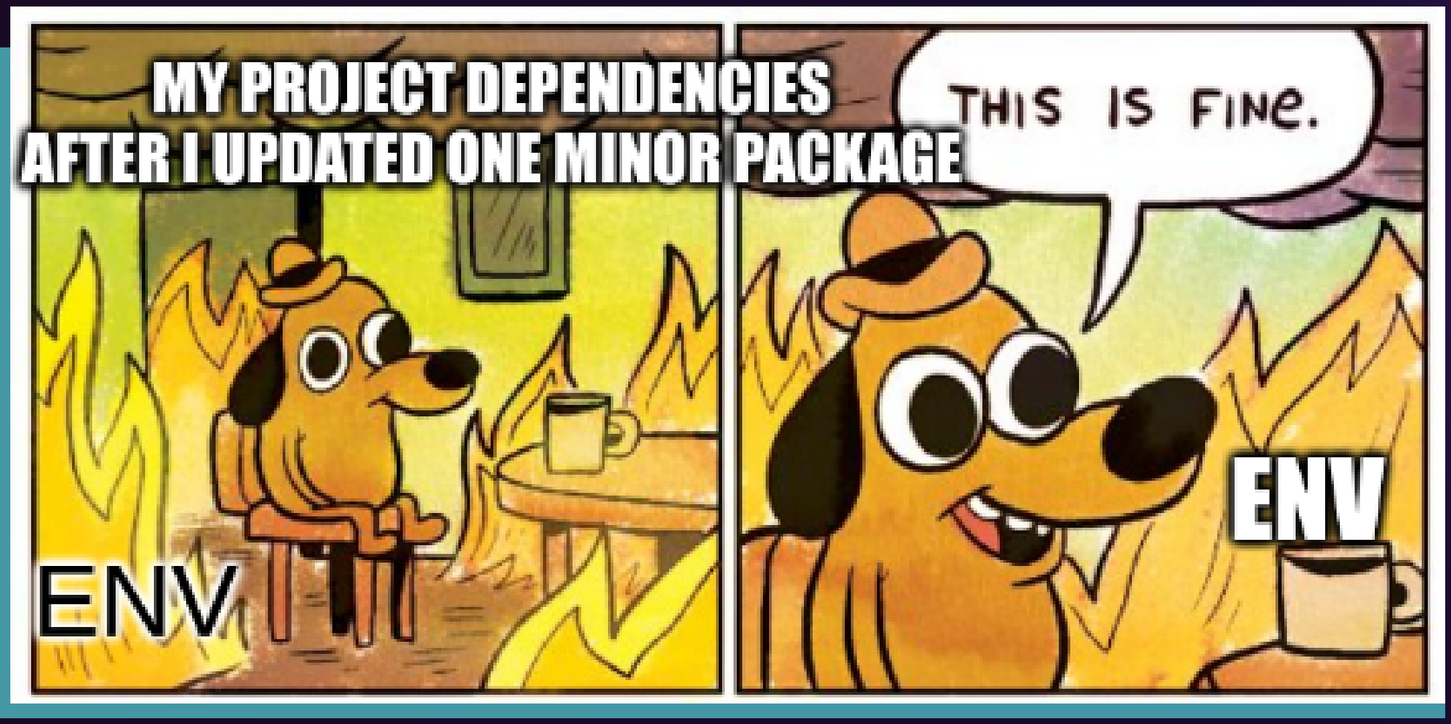
```
#!/usr/bin/env bash
set -euo pipefail

echo "=============================="
echo "🍥 FreeBSD QEMU Setup Script"
echo "=============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/releases/
ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")

  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install
manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using
$manager..."
      ${PKG_MANAGERS[$manager]}
"${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "🔧 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): "
CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): "
USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] &&
CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): "
USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] &&
RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default:
$DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] &&
DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load
kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi
  fi

  if [ ! -e /dev/kvm ]; then
    echo "✅ KVM is enabled."
  else
    echo "❌ KVM not available. Enable in BIOS/UEFI"
    exit 1
  fi

# ---------- MAIN WORKFLOW ----------
customize_resources
check_dependencies
check_kvm

echo "✅ Environment ready. Continue with ISO
```

```c
#include <stdio.h>

int main()
{
  int fahr;

  for (fahr = 300; fahr >= 0; fahr = fahr - 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
```


```
#!/usr/bin/env bash
set -euo pipefail

echo "=============================="
echo "🍥 FreeBSD QEMU Setup Script"
echo "=============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/releases/ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")

  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using $manager..."
      ${PKG_MANAGERS[$manager]} "${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "📌 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): " CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): " USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] && CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): " USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] && RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default: $DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] && DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi
  fi

  if [ -e /dev/kvm ]; then
    echo "✅ KVM acceleration can be used"
  else
    echo "❌ KVM is not available. Ensure BIOS/UEFI virtualization is enabled."
    exit 1
  fi
}

# ---------- MAIN ----------
customize_resources
check_dependencies
check_kvm

echo "✅ Environment ready. Continue with ISO
```

is a docker image is reproducible after certain amount of time like after 3 months

```
#include <stdio.h>

int main()
{
  int fahr;

  for (fahr = 300; fahr >= 0; fahr = fahr - 20)
    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
```



Problem with the global package manager

apt

snap

pacman

```
#!/usr/bin/env bash
set -euo pipefail

echo "=============================="
echo "🧠 FreeBSD QEMU Setup Script"
echo "=============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/releases/ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")

  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using $manager..."
      ${PKG_MANAGERS[$manager]} "${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "📌 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): " CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): " USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] && CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): " USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] && RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default: $DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] && DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi
  fi

if [ -e /dev/kvm ]; then
  ec
else
  ec
  ex
fi

----------
stom
eck_
eck_

ho "✅ Environment ready. Continue with ISO
```

```c
#include <stdio.h>

int main()
{
  int fahr;

  for (fahr = 300; fahr >= 0; fahr = fahr - 20)
    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
```

apt is the command-line tool that orchestrates the entire process of finding, downloading, and installing software packages and their dependencies on a individual system.

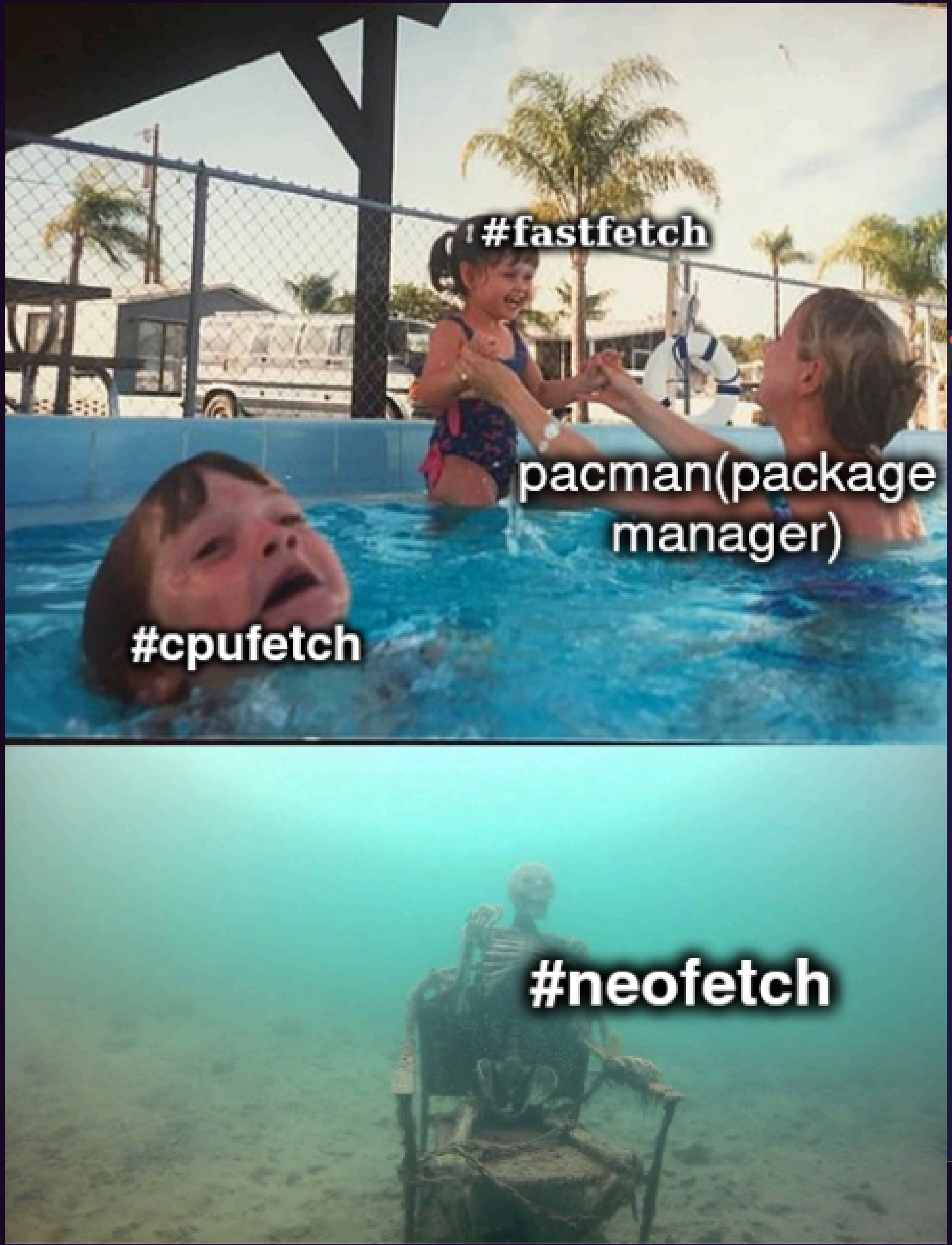
```
APT
├── Dependency Resolution
│   ├── Repositories (/etc/apt/sources.list)
│   └── Download Management
│       ├── HTTP/HTTPS
│       └── Local Cache (/var/cache/apt/archives)
├── The Backend
│   └── dpkg
└── System Integration
    ├── System Files
    └── Status Database (/var/lib/dpkg/status)
```

```
Flox
├── Environment Management
│   ├── Flox Hub (Remote Registry)
│   └── Nix Store (Local Package Cache)
├── The Backend
│   └── Nix (Declarative Build System)
└── User Integration
    ├── Command-line Interface
    └── User's PATH
```

```bash
#!/usr/bin/env bash
set -euo pipefail

echo "=============================="
echo "🧠 FreeBSD QEMU Setup Script"
echo "=============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/releases/ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")

  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using $manager..."
      ${PKG_MANAGERS[$manager]} "${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "📌 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "  ➤ CPUs : $CPUS"
  echo "  ➤ RAM  : $RAM MB"
  echo "  ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): " CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): " USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] && CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): " USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] && RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default: $DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] && DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "  ➤ CPUs : $CPUS"
  echo "  ➤ RAM  : $RAM MB"
  echo "  ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi
    if [ ! -e /dev/kvm ]; then
      echo "❌ KVM still not available. Check BIOS/UEFI."
      exit 1
    fi
  fi
}

# ---------- MAIN ----------
check_dependencies
customize_resources
check_kvm

echo "✅ Environment ready. Continue with ISO..."
```

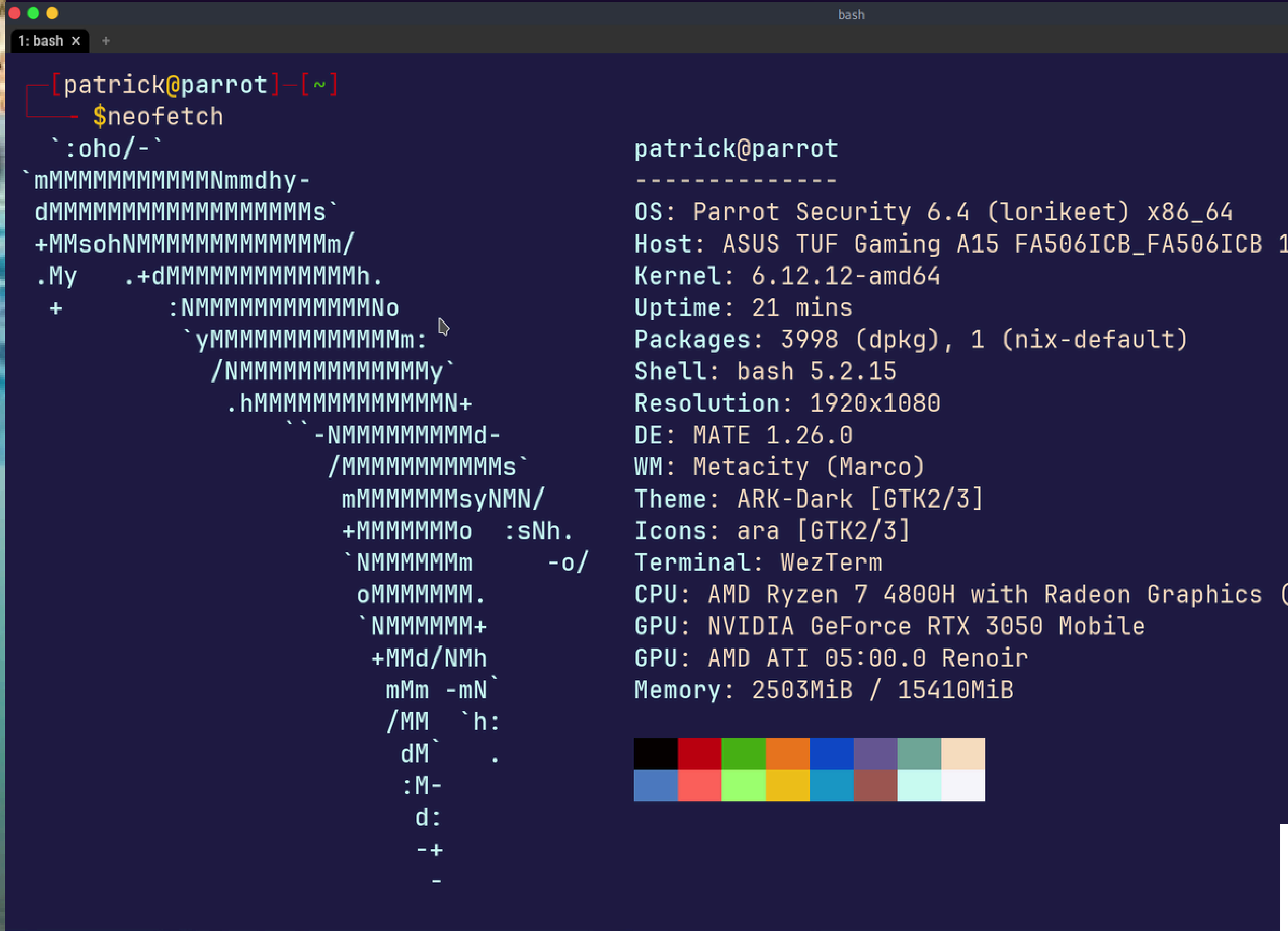
FOSS UNITED

```c
#include <stdio.h>
```



neofetch binary had been removed from pacman(arch) package manager

```bash
    echo -n "🌐 Checking internet... "
    if ping -c 1 google.com &>/dev/null; then
        echo "connected."
    else
        echo "❌ No internet connection. Please install manually: ${missing_tools[*]}"
        exit 1
    fi

    local pkg_found=false
    for manager in "${!PKG_MANAGERS[@]}"; do
        if command -v "$manager" &>/dev/null; then
            echo "📦 Installing missing packages using $manager..."
            ${PKG_MANAGERS[$manager]} "${missing_tools[@]}"
            pkg_found=true
            break
        fi
    done

    if ! $pkg_found; then
        echo "❌ No supported package manager found."
        echo "🛠 Please install manually: ${missing_tools[*]}"
        exit 1
    fi
}
```




```bash
local pkg_found=false
for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
        echo "📦 Installing missing packages using
```


FOSS UNITED

```
#include <stdio.h>

int main()
{
  int fahr;

  for (fahr = 300; fahr
>= 0; fahr = fahr - 20)
        printf("%3
%6.1f\n",      fahr,
(5.0/9.0)*(fahr-32));
```

Your code's personal bubble

```
#!/usr/bin/env bash
set -euo pipefail

echo "=============================="
echo "🍩 FreeBSD QEMU Setup Script"
echo "=============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/rele
ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")
```

1. Write once, runs anywhere

2. Self contained and reliable

3 .Know what's inside

```
  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install
manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using
$manager..."
      ${PKG_MANAGERS[$manager]}
"${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "📌 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): "
CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): "
USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] &&
CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): "
USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] &&
RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default:
$DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] &&
DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load
kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi
```

LET"S DIVE INTO FLOX {{BEHIND THE HOOD}}

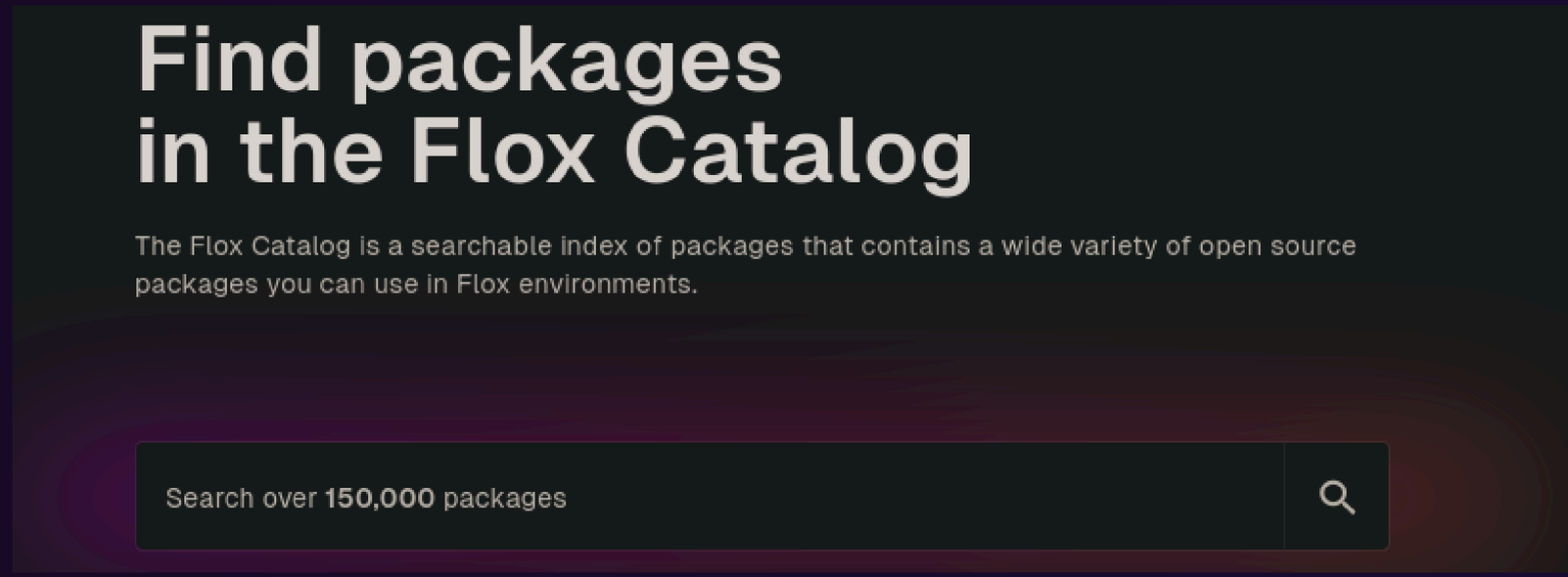
FOSS UNITED

CORE Feature of FLOX

```
{
  int fahr;

  for (fahr = 300; fahr >= 0; fahr = fahr - 20)
      printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
```

```
$ flox cli [or.....] flox shell
```

```
$ flox package manager(floxhub)
```




```
#!/usr/bin/env bash
set -euo pipefail

echo "=============================="
echo "🟤 FreeBSD QEMU Setup Script"
echo "=============================="

# ---------- CONFIGURATION ----------
DISK_FILE="freebsd.qcow2"
DISK_SIZE="20G"
RAM="4096"
CPUS="4"
BASE_URL="https://download.freebsd.org/ftp/releases/ISO-IMAGES"
ISO_DIR="isos"
mkdir -p "$ISO_DIR"

# ---------- DECLARATIVE VARIABLES ----------
REQUIRED_TOOLS=(
  curl
  wget
  xz
  qemu-img
  qemu-system-x86_64
)

# Mapping of package managers to install commands
declare -A PKG_MANAGERS=(
  [apt]="sudo apt update && sudo apt install -y"
  [dnf]="sudo dnf install -y"
  [pacman]="sudo pacman -Sy --noconfirm"
  [pkg]="sudo pkg install -y"
  [nix-env]="nix-env -iA nixpkgs"
)

# ---------- FUNCTIONS ----------
check_dependencies() {
  echo "🔍 Checking for required tools..."
  MISSING_TOOLS=()

  for tool in "${REQUIRED_TOOLS[@]}"; do
    if ! command -v "$tool" &>/dev/null; then
      MISSING_TOOLS+=("$tool")
    fi
  done

  if [ ${#MISSING_TOOLS[@]} -eq 0 ]; then
    echo "✅ All dependencies are installed."
    return 0
  fi

  echo "⚠️ Missing: ${MISSING_TOOLS[*]}"
  install_dependencies "${MISSING_TOOLS[@]}"
}

install_dependencies() {
  local missing_tools=("$@")

  echo -n "🌐 Checking internet... "
  if ping -c 1 google.com &>/dev/null; then
    echo "connected."
  else
    echo "❌ No internet connection. Please install manually: ${missing_tools[*]}"
    exit 1
  fi

  local pkg_found=false
  for manager in "${!PKG_MANAGERS[@]}"; do
    if command -v "$manager" &>/dev/null; then
      echo "📦 Installing missing packages using $manager..."
      ${PKG_MANAGERS[$manager]} "${missing_tools[@]}"
      pkg_found=true
      break
    fi
  done

  if ! $pkg_found; then
    echo "❌ No supported package manager found."
    echo "🛠 Please install manually: ${missing_tools[*]}"
    exit 1
  fi
}

customize_resources() {
  echo ""
  echo " Default Resources:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"

  read -p "✨ Customize CPU/RAM/Disk? (y/N): " CUSTOMIZE
  CUSTOMIZE=${CUSTOMIZE,,}

  if [[ "$CUSTOMIZE" == "y" ]]; then
    read -p "Enter CPU cores (default: $CPUS): " USER_CPUS
    [[ "$USER_CPUS" =~ ^[0-9]+$ ]] && CPUS="$USER_CPUS"

    read -p "Enter RAM in MB (default: $RAM): " USER_RAM
    [[ "$USER_RAM" =~ ^[0-9]+$ ]] && RAM="$USER_RAM"

    read -p "Enter disk size (e.g., 25G) (default: $DISK_SIZE): " USER_DISK
    [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] && DISK_SIZE="$USER_DISK"
  fi

  echo ""
  echo "🚀 Final Configuration:"
  echo "   ➤ CPUs : $CPUS"
  echo "   ➤ RAM  : $RAM MB"
  echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
  echo "🔍 Checking KVM support..."

  if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
    echo "✅ CPU supports virtualization."
  else
    echo "❌ No virtualization support detected. Exiting."
    exit 1
  fi

  if [ ! -e /dev/kvm ]; then
    echo "⚠️ /dev/kvm not found. Attempting to load kernel modules..."
    if grep -q vmx /proc/cpuinfo; then
      sudo modprobe kvm_intel || true
    elif grep -q svm /proc/cpuinfo; then
      sudo modprobe kvm_amd || true
    fi

    if [ -e /dev/kvm ]; then
      echo "✅ KVM modules loaded successfully."
    else
      echo "❌ KVM still not available. Check BIOS/UEFI."
      exit 1
    fi
  fi
}

# ---------- MAIN ----------
customize_resources
check_dependencies
check_kvm

echo "✅ Environment ready. Continue with ISO..."
```

Mr. Tech Burger

Distro {Arch, Ubuntu,298++}

FLOX

our awesome project

Global ENV {variables}

kernel{linux, OS x, Freebsd } and architecture [x86, arm ,risc -v]

The shell(you again?)

```
$ git init testdir && cd testdir
$ flox init
```

```
$ flox install neovim
$ flox activate (isolation starts here)
```

SHELL-->

FOSS UNITED

The shell(you again?)

```
{
    int fahr;

    for (fahr = 300; fahr >= 0; fahr = fahr - 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

```
$ git init testdir && cd testdir
$ flox init
```

```
$ flox install neovim
$ flox activate (isolation starts here)
```

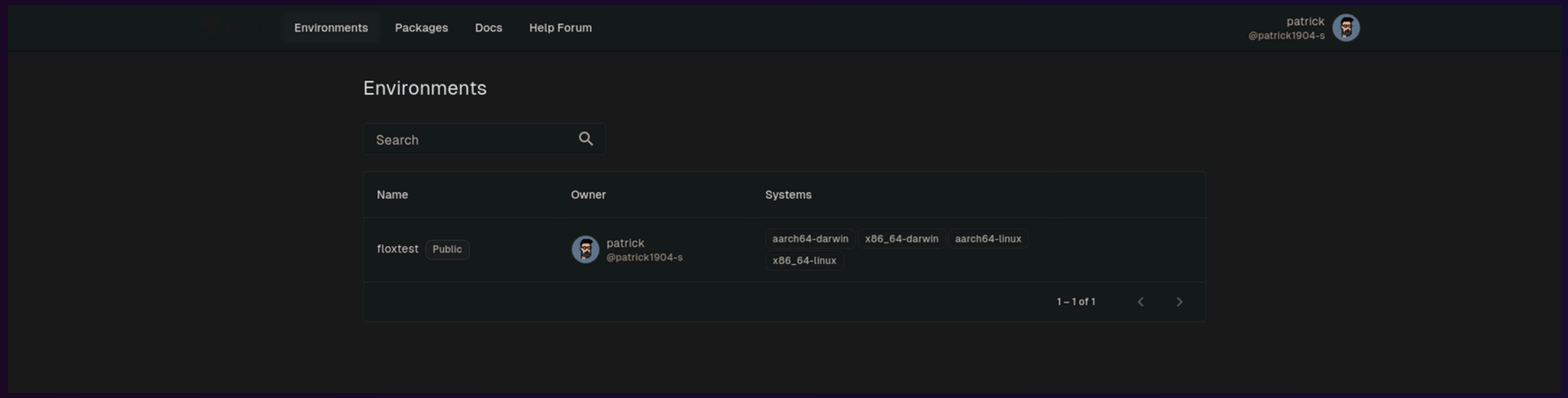
SHELL-->

SUB (SHELL)

FOSS UNITED

Share YOur ENvironment with anyone

```
$ flox push
✅  example-project successfully pushed to FloxHub

 Use 'flox pull youruser/example-project' to get this environment in any other location.
```



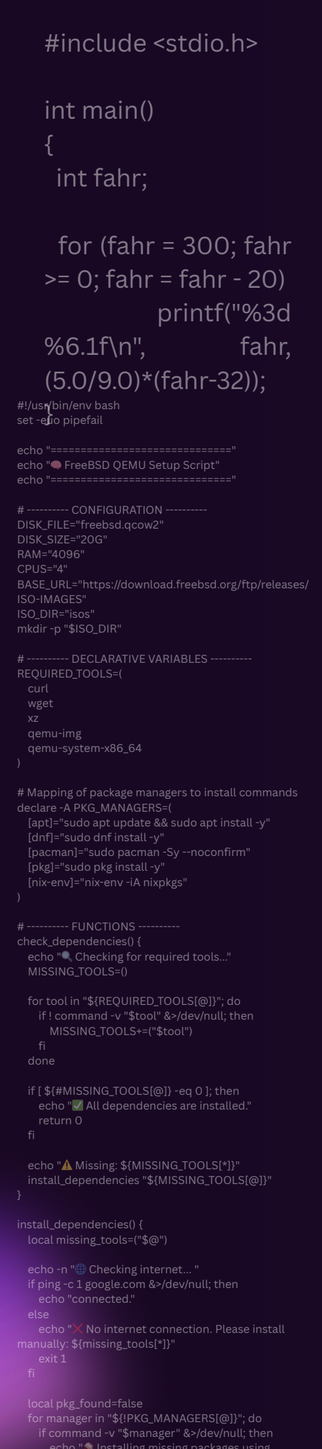
floxhub

FOSS UNITED

Heart OF the FLOX ENV

manifest.toml

The manifest.toml file is the cornerstone of Environment as Code, acting as a blueprint that guarantees your project's environment is consistent, reproducible, and portable. It turns "it works on my machine" into "it works everywhere.



flox

1: flox ×

GNU nano 7.2 /home/patrick/.cache/flox/process/.tmpbUXLhF/manifest.3Psm6O.toml

```
##   _Everything_ you need to know about the _manifest_ is here:
##
##               https://flox.dev/docs/concepts/manifest
##
## -------------------------------------------------------------
# Flox manifest version managed by Flox CLI
version = 1


## Install Packages ---------------------------------------------
##  $ flox install gum  <- puts a package in [install] section below
##  $ flox search gum   <- search for a package
##  $ flox show gum     <- show all versions of a package
## -------------------------------------------------------------
[install]
# gum.pkg-path = "gum"
# gum.version = "^0.14.5"


## Environment Variables ----------------------------------------
##  ... available for use in the activated environment
##      as well as [hook], [profile] scripts and [services] below.
## -------------------------------------------------------------
[vars]
# INTRO_MESSAGE = "It's gettin' Flox in here"


## Activation Hook ----------------------------------------------
##  ... run by _bash_ shell when you run 'flox activate'.
## -------------------------------------------------------------
[hook]
# on-activate = '''
#   # -> Set variables, create files and directories
#   # -> Perform initialization steps, e.g. create a python venv
#   # -> Useful environment variables:
#   #      - FLOX_ENV_PROJECT=/home/user/example
#   #      - FLOX_ENV=/home/user/example/.flox/run
#   #      - FLOX_ENV_CACHE=/home/user/example/.flox/cache
# '''
```

[line 3/104 (2%), col 1/65 (1%), char 74/3415 (2%)]

^H Help	^O Read File	^R Replace	^V Paste	^G Go To Line	^Y Redo	M-6 Copy	^Q Where Was	M-W Next	▸ Forward	^ Next Word
^X Exit	^F Where Is	^K Cut	^T Execute	^Z Undo	M-A Set Mark	M-] To Bracket	M-Q Previous	^B Back	^◂ Prev Word	^ Home

Let’s mess with ENV (WORKSHOP)

Github

```
    echo -n "🌐 Checking internet... "
    if ping -c 1 google.com &>/dev/null; then
        echo "connected."
    else
        echo "❌ No internet connection. Please install
manually: ${missing_tools[*]}"
        exit 1
    fi

    local pkg_found=false
    for manager in "${!PKG_MANAGERS[@]}"; do
        if command -v "$manager" &>/dev/null; then
            echo "📦 Installing missing packages using
$manager..."
            ${PKG_MANAGERS[$manager]}
"${missing_tools[@]}"
            pkg_found=true
            break
        fi
    done

    if ! $pkg_found; then
        echo "❌ No supported package manager found."
        echo "🔧 Please install manually: ${missing_tools[*]}"
        exit 1
    fi
}

customize_resources() {
    echo ""
    echo " Default Resources:"
    echo "   ➤ CPUs : $CPUS"
    echo "   ➤ RAM  : $RAM MB"
    echo "   ➤ Disk : $DISK_SIZE"

    read -p "✨ Customize CPU/RAM/Disk? (y/N): "
CUSTOMIZE
    CUSTOMIZE=${CUSTOMIZE,,}

    if [[ "$CUSTOMIZE" == "y" ]]; then
        read -p "Enter CPU cores (default: $CPUS): "
USER_CPUS
        [[ "$USER_CPUS" =~ ^[0-9]+$ ]] &&
CPUS="$USER_CPUS"

        read -p "Enter RAM in MB (default: $RAM): "
USER_RAM
        [[ "$USER_RAM" =~ ^[0-9]+$ ]] &&
RAM="$USER_RAM"

        read -p "Enter disk size (e.g., 25G) (default:
$DISK_SIZE): " USER_DISK
        [[ "$USER_DISK" =~ ^[0-9]+[GgMm]?$ ]] &&
DISK_SIZE="$USER_DISK"
    fi

    echo ""
    echo "🚀 Final Configuration:"
    echo "   ➤ CPUs : $CPUS"
    echo "   ➤ RAM  : $RAM MB"
    echo "   ➤ Disk : $DISK_SIZE"
}

check_kvm() {
    echo "🔍 Checking KVM support..."

    if grep -E -q '(vmx|svm)' /proc/cpuinfo; then
        echo "✅ CPU supports virtualization."
    else
        echo "❌ No virtualization support detected. Exiting."
        exit 1
    fi

    if [ ! -e /dev/kvm ]; then
        echo "⚠️ /dev/kvm not found. Attempting to load
kernel modules..."
        if grep -q vmx /proc/cpuinfo; then
            sudo modprobe kvm_intel || true
        elif grep -q svm /proc/cpuinfo; then
            sudo modprobe kvm_amd || true
        fi

    if
        echo "✅
    el
        KVM not
        1
    fi
}

# ---
customize_resources
check_kvm
check_dependencies

echo "✅ Environment ready. Continue with ISO
```

FOSS UNITED

Reference

1. **FLox installation : https://flox.dev/docs/install-flox/install/**
2. **Flox Documentation : https://flox.dev/docs/**
3. **Awesomeness of nix : https://github.com/nix-community/awesome-nix**

Add up your suggetion