Enhance and integrate with ultra-modern /state of theanimations , emojis, colocolors #!/usr/bin/env zsh

```
# ==============================================================================
# QUANTUM NEXUS DEEP CLEAN SYSTEM v3.0 - AI-Powered Deep Detection & Purge
# ==============================================================================
# Features: Quantum header, 7-layer detection, atomic operations, deep file
# scanning, malware detection, orphan cleanup, performance optimization
# ==============================================================================

# --- 0. QUANTUM GLOBALS & SETUP ---
export QNEXUS_ROOT="${HOME}/.quantum_nexus"
export QNEXUS_LOGS="${QNEXUS_ROOT}/logs"
export QNEXUS_DATA="${QNEXUS_ROOT}/data"
export QNEXUS_CACHE="${QNEXUS_ROOT}/cache"
export QNEXUS_QUARANTINE="${QNEXUS_ROOT}/quarantine"
export QNEXUS_SNAPSHOTS="${QNEXUS_ROOT}/snapshots"

# Create quantum directories
mkdir -p "${QNEXUS_LOGS}" "${QNEXUS_DATA}" "${QNEXUS_CACHE}" \
        "${QNEXUS_QUARANTINE}" "${QNEXUS_SNAPSHOTS}" 2>/dev/null

export QNEXUS_LOG="${QNEXUS_LOGS}/quantum_$(date +%Y%m%d_%H%M%S).log"

# Quantum color system
typeset -gA Q_COLORS=(
    [grad1]="\033[38;5;39m"    [grad2]="\033[38;5;45m"    [grad3]="\033[38;5;51m"
    [grad4]="\033[38;5;87m"    [grad5]="\033[38;5;129m"   [grad6]="\033[38;5;165m"
    [success]="\033[38;5;46m"  [error]="\033[38;5;196m"   [warning]="\033[38;5;226m"
    [info]="\033[38;5;51m"     [debug]="\033[38;5;245m"   [reset]="\033[0m"
)

# System detection
export SYSTEM_ARCH=$(uname -m)
export SYSTEM_OS=$(uname -s)
export SYSTEM_VERSION=$(sw_vers -productVersion 2>/dev/null || uname -r)
export IS_MACOS=$([[ "$SYSTEM_OS" == "Darwin" ]] && echo "true" || echo "false")
export IS_ARM=$([[ "$SYSTEM_ARCH" =~ "arm64|aarch64" ]] && echo "true" || echo "false")

# Quantum logging
quantum_log() {
    local timestamp=$(date "+%Y-%m-%d %H:%M:%S.%3N")
    local level="$1"
    local message="$2"
    echo "[$timestamp] [$level] $message" >> "$QNEXUS_LOG"
}

# --- 1. QUANTUM HEADER & VISUALS ---
quantum_header() {
    clear
    local cols=$(tput cols 2>/dev/null || echo 80)

    # Quantum rainbow palette
    local rainbow=(196 202 208 214 220 190 154 118 82 46 51 39 33 27 93 129 165 201 200 198 197)

    # Quantum animation frames
    local quantum_frames=(
        "⠋" "⠙" "⠹" "⠸" "⠼" "⠴" "⠦" "⠧" "⠇" "⠏"
        "🔄" "⚡" "🌀" "🌪" "✨" "💫" "🌟" "🌠" "🌌" "🌠"
    )

    # Get system telemetry
    local cpu_temp=$(sudo powermetrics --samplers smc -n1 2>/dev/null | grep -i "cpu die" | head -1 | awk
'{print $4}' || echo "N/A")
    local mem_used=$(memory_pressure 2>/dev/null | grep "System-wide memory free percentage:" | awk
'{print 100 - $5}' || echo "0")
    local disk_used=$(df -h / 2>/dev/null | awk 'NR==2 {print $5}' | sed 's/%//' || echo "0")
    local net_activity=$(netstat -an 2>/dev/null | grep -c ESTABLISHED || echo "0")

    # Calculate quantum health score
    local health_score=$(awk -v cpu="$cpu_temp" -v mem="$mem_used" -v disk="$disk_used" -v
net="$net_activity" '
        BEGIN {
            cpu_score = cpu == "N/A" ? 90 : (cpu > 80 ? 30 : (cpu > 60 ? 70 : 95));
            mem_score = 100 - mem;
            disk_score = 100 - disk;
            net_score = net > 100 ? 60 : (net > 50 ? 80 : 95);
            total = (cpu_score * 0.3) + (mem_score * 0.3) + (disk_score * 0.2) + (net_score * 0.2);
            printf "%.0f", total;
        }')

    # Sparkline generator
    quantum_sparkline() {
        local values=(${(@s:,:)1})
        local blocks=(▁ ▂ ▃ ▄ ▅ ▆ ▇ █)
        local output=""
        for v in $values; do
            local idx=$(( (v * 7) / 100 ))
            (( idx > 7 )) && idx=7
            (( idx < 0 )) && idx=0
            local color_idx=$(( (RANDOM % 7) + 1 ))
            output+="${Q_COLORS[grad$color_idx]}${blocks[$idx]}"
        done
        echo -n "$output${Q_COLORS[reset]}"
    }

    # Generate telemetry sparkline
    local spark_values=($health_score $mem_used $disk_used $net_activity)
    local sparkline=$(quantum_sparkline ${(j:,:)spark_values})

    # Quantum header
    echo -e "${Q_COLORS[grad1]}"
    echo
" ╔══════════════════════════════════════════════════════════════════════════════
══════════════════════════════════════════════════════════════╗ "
    echo " ║                    🔬 QUANTUM NEXUS DEEP SCAN v3.0 🔬                    ║ "
    echo
" ╠══════════════════════════════════════════════════════════════════════════════
══════════════════════════════════════════════════════════════╣ "

    # Animated status line
    local frame_idx=$(( (SECONDS % ${#quantum_frames}) + 1 ))
    local anim_frame="${quantum_frames[$frame_idx]}"

    echo -e " ║   ${Q_COLORS[grad3]}${anim_frame} LIVE TELEMETRY: ${Q_COLORS[grad6]}CPU: $
```

```
{cpu_temp}°C  RAM: ${mem_used}%  DISK: ${disk_used}%  CONN: ${net_activity}"
    echo -e "║  ${Q_COLORS[grad2]}🏥 SYSTEM HEALTH: ${Q_COLORS[grad4]}${health_score}/100  $
{sparkline}"

    # Quantum scanning animation
    echo -n "║  ${Q_COLORS[grad5]}🔄 QUANTUM SCAN: "
    local scan_width=40
    local scan_pos=$(( (SECONDS * 2) % scan_width ))
    for ((i=0; i<scan_width; i++)); do
        if [[ $i -eq $scan_pos ]]; then
            echo -n "${Q_COLORS[grad6]}█"
        elif [[ $i -lt $scan_pos ]]; then
            echo -n "${Q_COLORS[grad3]}─"
        else
            echo -n "${Q_COLORS[grad1]}▒"
        fi
    done
    echo -e "${Q_COLORS[reset]}"

    echo
"╚═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════════════════════╝"
    echo -e "${Q_COLORS[reset]}"

    quantum_log "INFO" "Quantum header displayed - Health: $health_score"
}

# --- 2. QUANTUM DEEP DETECTION SYSTEM ---
quantum_deep_detect() {
    quantum_header
    echo -e "${Q_COLORS[grad1]}[🔍] QUANTUM DEEP DETECTION INITIATED${Q_COLORS[reset]}"
    echo

    # Layer 1: System Configuration Detection
    echo -e "${Q_COLORS[grad2]}━━━━━━━ LAYER 1: SYSTEM CONFIGURATION ━━━━━━━$
{Q_COLORS[reset]}"
    quantum_detect_system_config
    echo

    # Layer 2: Application & Binary Detection
    echo -e "${Q_COLORS[grad3]}━━━━━━━ LAYER 2: APPLICATION SIGNATURES ━━━━━━━$
{Q_COLORS[reset]}"
    quantum_detect_applications
    echo

    # Layer 3: Malware & Anomaly Detection
    echo -e "${Q_COLORS[grad4]}━━━━━━━ LAYER 3: THREAT DETECTION ━━━━━━━${Q_COLORS[reset]}"
    quantum_detect_threats
    echo

    # Layer 4: Performance & Resource Analysis
    echo -e "${Q_COLORS[grad5]}━━━━━━━ LAYER 4: PERFORMANCE METRICS ━━━━━━━$
{Q_COLORS[reset]}"
    quantum_analyze_performance
    echo
    # Layer 5: Orphan & Dead Data Detection
    echo -e "${Q_COLORS[grad6]}━━━━━━━ LAYER 5: ORPHAN DATA DETECTION ━━━━━━━$
{Q_COLORS[reset]}"
    quantum_detect_orphans
    echo

    # Layer 6: Permission & Security Analysis
    echo -e "${Q_COLORS[grad1]}━━━━━━━ LAYER 6: SECURITY AUDIT ━━━━━━━${Q_COLORS[reset]}"
    quantum_audit_security
    echo

    # Layer 7: Integration & Dependency Mapping
    echo -e "${Q_COLORS[grad2]}━━━━━━━ LAYER 7: DEPENDENCY GRAPH ━━━━━━━$
{Q_COLORS[reset]}"
    quantum_map_dependencies
    echo

    quantum_log "INFO" "7-layer deep detection completed"
}

quantum_detect_system_config() {
    echo -e "${Q_COLORS[info]}Scanning system configuration...${Q_COLORS[reset]}"

    # Detect shell configurations
    local shell_configs=(
        ~/.zshrc ~/.bashrc ~/.zprofile ~/.bash_profile
        ~/.zshenv ~/.zlogin ~/.zlogout ~/.profile
        ~/.config/zsh ~/.config/bash ~/.oh-my-zsh
    )

    local config_count=0
    local total_size=0
    for config in $shell_configs; do
        if [[ -f "$config" ]] || [[ -d "$config" ]]; then
            local size=$(du -sk "$config" 2>/dev/null | cut -f1 || echo 0)
            (( total_size += size ))
            (( config_count++ ))
            echo -e "  ${Q_COLORS[success]}✓${Q_COLORS[reset]} Found: $(basename $config) (${size}KB)"
        fi
    done

    # Detect environment variables
    local env_vars=$(env | wc -l)
    local path_entries=$(echo $PATH | tr ':' '\n' | wc -l)

    echo -e "  ${Q_COLORS[info]}Configuration Summary:${Q_COLORS[reset]}"
    echo -e "    • Config files: $config_count"
    echo -e "    • Total size: $((total_size / 1024))MB"
    echo -e "    • Environment variables: $env_vars"
    echo -e "    • PATH entries: $path_entries"

    # Detect conflicting configurations
    quantum_detect_conflicts
}

quantum_detect_conflicts() {
    echo -e "${Q_COLORS[info]}Checking for configuration conflicts...${Q_COLORS[reset]}"
    # Check for duplicate PATH entries
    local duplicate_paths=$(echo $PATH | tr ':' '\n' | sort | uniq -d)
    if [[ -n "$duplicate_paths" ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ Duplicate PATH entries found${Q_COLORS[reset]}"
        echo "$duplicate_paths" | while read -r path; do
            echo -e "    • $path"
        done
    fi

    # Check for conflicting aliases
    if [[ -f ~/.zshrc ]]; then
        local aliases=$(grep -E '^alias ' ~/.zshrc | cut -d' ' -f2 | cut -d'=' -f1 | sort | uniq -d)
        if [[ -n "$aliases" ]]; then
            echo -e "  ${Q_COLORS[warning]}⚠ Conflicting aliases detected${Q_COLORS[reset]}"
        fi
    fi
}

quantum_detect_applications() {
    echo -e "${Q_COLORS[info]}Detecting installed applications...${Q_COLORS[reset]}"

    # Homebrew detection
    if command -v brew &> /dev/null; then
        local brew_count=$(brew list --formula | wc -l | tr -d ' ')
        local cask_count=$(brew list --cask | wc -l | tr -d ' ')
        echo -e "  ${Q_COLORS[success]}✓${Q_COLORS[reset]} Homebrew detected: $brew_count formulae,
$cask_count casks"

        # Detect outdated packages
        local outdated=$(brew outdated --quiet | wc -l | tr -d ' ')
        if [[ $outdated -gt 0 ]]; then
            echo -e "  ${Q_COLORS[warning]}⚠ $outdated outdated packages${Q_COLORS[reset]}"
        fi
    fi

    # Node.js detection
    if command -v node &> /dev/null; then
        local node_version=$(node --version)
        local npm_packages=$(npm list -g --depth=0 2>/dev/null | wc -l)
        echo -e "  ${Q_COLORS[success]}✓${Q_COLORS[reset]} Node.js $node_version ($((npm_packages -
1)) global packages)"
    fi

    # Python detection
    if command -v python3 &> /dev/null; then
        local python_version=$(python3 --version 2>&1)
        local pip_packages=$(pip3 list --format=freeze 2>/dev/null | wc -l)
        echo -e "  ${Q_COLORS[success]}✓${Q_COLORS[reset]} $python_version ($pip_packages
packages)"
    fi

    # Ruby detection
    if command -v ruby &> /dev/null; then
        local ruby_version=$(ruby --version | cut -d' ' -f2)
        local gem_count=$(gem list --local | wc -l)
        echo -e "  ${Q_COLORS[success]}✓${Q_COLORS[reset]} Ruby $ruby_version ($gem_count gems)"
    fi

    # Docker detection
    if command -v docker &> /dev/null; then
        local docker_containers=$(docker ps -aq 2>/dev/null | wc -l)
        local docker_images=$(docker images -q 2>/dev/null | wc -l)
        echo -e "  ${Q_COLORS[success]}✓${Q_COLORS[reset]} Docker ($docker_containers containers,
$docker_images images)"
    fi

    # Count total binaries in PATH
    local total_binaries=$(find ${(ps.:.)PATH} -type f -executable 2>/dev/null | wc -l)
    echo -e "  ${Q_COLORS[info]}Total executables in PATH: $total_binaries${Q_COLORS[reset]}"
}

quantum_detect_threats() {
    echo -e "${Q_COLORS[info]}Scanning for threats and anomalies...${Q_COLORS[reset]}"

    # Check for suspicious files
    local suspicious_locations=(
        "$HOME/Library/LaunchAgents"
        "$HOME/Library/LaunchDaemons"
        "$HOME/.config/autostart"
        "/tmp"
        "/var/tmp"
    )

    local threat_count=0
    for location in $suspicious_locations; do
        if [[ -d "$location" ]]; then
            # Look for unusual permissions
            local suspicious=$(find "$location" -type f -perm /u=x,g=x,o=x 2>/dev/null | head -10)
            if [[ -n "$suspicious" ]]; then
                (( threat_count += $(echo "$suspicious" | wc -l) ))
                echo -e "  ${Q_COLORS[warning]}⚠ Executables in $location${Q_COLORS[reset]}"
            fi
        fi
    done

    # Check for known malware patterns
    if [[ "$IS_MACOS" == "true" ]]; then
        # macOS specific malware checks
        local malware_patterns=(
            "com.adobe.*"
            "com.google.*.agent"
            "*.mine"
            "cryptominer"
        )

        for pattern in $malware_patterns; do
            local matches=$(find /Applications ~/Applications -name "*$pattern*" -maxdepth 2 2>/dev/null)
            if [[ -n "$matches" ]]; then
                echo -e "  ${Q_COLORS[error]}✗ Potential malware pattern: $pattern${Q_COLORS[reset]}"
                echo "$matches" | while read -r match; do
                    echo -e "    • $match"
                done
            fi
        done
    fi

    # Check for cryptocurrency miners
    local miner_processes=$(ps aux | grep -i "miner\|xmrig\|ccminer\|cpuminer" | grep -v grep)
    if [[ -n "$miner_processes" ]]; then
        echo -e "  ${Q_COLORS[error]}✗ Cryptocurrency miners detected!${Q_COLORS[reset]}"
        echo "$miner_processes"
    fi

    if [[ $threat_count -eq 0 ]]; then
        echo -e "  ${Q_COLORS[success]}✓ No immediate threats detected${Q_COLORS[reset]}"
    fi
}

quantum_analyze_performance() {
    echo -e "${Q_COLORS[info]}Analyzing system performance...${Q_COLORS[reset]}"

    # CPU usage
    local cpu_usage=$(top -l 1 | grep "CPU usage" | awk '{print $3}' | sed 's/%//' || echo "0")

    # Memory usage
    local mem_total=$(sysctl -n hw.memsize 2>/dev/null | awk '{printf "%.2f", $1/1024/1024/1024}' || echo "0")
    local mem_stats=$(vm_stat 2>/dev/null | head -5)
    local mem_free_pages=$(echo "${mem_stats}" | grep "Pages free" | awk '{print $3}' | sed 's/\.//')
    local page_size=$(echo "${mem_stats}" | head -1 | awk '{print $8}')
    local mem_free_gb=$(awk "BEGIN {printf \"%.2f\", (${mem_free_pages} * ${page_size}) / 1024 / 1024 /
1024}")
    local mem_used_gb=$(awk "BEGIN {printf \"%.2f\", ${mem_total} - ${mem_free_gb}}")
    local mem_percent=$(awk "BEGIN {printf \"%.1f\", (${mem_used_gb} / ${mem_total}) * 100}")

    # Disk usage
    local disk_usage=$(df -h / 2>/dev/null | awk 'NR==2 {print $5}' | sed 's/%//' || echo "0")

    # Network connections
    local net_connections=$(netstat -an 2>/dev/null | grep -c ESTABLISHED || echo "0")

    # Performance score calculation
    local perf_score=$(awk -v cpu="$cpu_usage" -v mem="$mem_percent" -v disk="$disk_usage" -v
net="$net_connections" '
        BEGIN {
            cpu_score = 100 - cpu;
            mem_score = 100 - mem;
            disk_score = 100 - disk;
            net_score = net > 100 ? 60 : (net > 50 ? 80 : 95);
            total = (cpu_score * 0.3) + (mem_score * 0.3) + (disk_score * 0.2) + (net_score * 0.2);
            printf "%.0f", total;
        }')

    echo -e "  ${Q_COLORS[info]}Performance Metrics:${Q_COLORS[reset]}"
    echo -e "    • CPU Usage: ${cpu_usage}%"
    echo -e "    • Memory Usage: ${mem_percent}% (${mem_used_gb}GB / ${mem_total}GB)"
    echo -e "    • Disk Usage: ${disk_usage}%"
    echo -e "    • Network Connections: ${net_connections}"
    echo -e "    • Performance Score: ${perf_score}/100"
    # Identify performance bottlenecks
    if [[ $cpu_usage -gt 80 ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ High CPU usage detected${Q_COLORS[reset]}"
    fi

    if [[ $mem_percent -gt 85 ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ High memory usage detected${Q_COLORS[reset]}"
    fi

    if [[ $disk_usage -gt 90 ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ High disk usage detected${Q_COLORS[reset]}"
    fi
}

quantum_detect_orphans() {
    echo -e "${Q_COLORS[info]}Detecting orphaned files and directories...${Q_COLORS[reset]}"

    # Common orphan locations
    local orphan_locations=(
        "$HOME/.cache"
        "$HOME/.local/share/Trash"
        "$HOME/Library/Caches"
        "$HOME/Library/Logs"
        "$HOME/Library/Saved Application State"
        "/tmp"
        "/var/tmp"
    )

    local total_orphans=0
    local total_size=0

    for location in $orphan_locations; do
        if [[ -d "$location" ]]; then
            local orphan_count=$(find "$location" -type f -atime +30 2>/dev/null | wc -l)
            local orphan_size=$(find "$location" -type f -atime +30 -exec du -sk {} + 2>/dev/null | awk '{sum+=$1}
END {print sum}')

            (( total_orphans += orphan_count ))
            (( total_size += orphan_size ))

            if [[ $orphan_count -gt 0 ]]; then
                echo -e "  ${Q_COLORS[warning]}⚠ $orphan_count orphans in $(basename $location) ($
{orphan_size}KB)${Q_COLORS[reset]}"
            fi
        fi
    done

    # Detect orphaned symlinks
    local broken_links=$(find "$HOME" -type l ! -exec test -e {} \; -print 2>/dev/null | wc -l)
    if [[ $broken_links -gt 0 ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ $broken_links broken symlinks detected${Q_COLORS[reset]}"
    fi

    echo -e "  ${Q_COLORS[info]}Total orphaned data: $total_orphans files (${total_size}KB)$
{Q_COLORS[reset]}"
}
```

```
quantum_audit_security() {
    echo -e "${Q_COLORS[info]}Conducting security audit...${Q_COLORS[reset]}"

    # Check file permissions in home directory
    local world_writable=$(find "$HOME" -type f -perm -o+w ! -path "*/\.*" 2>/dev/null | head -20)
    if [[ -n "$world_writable" ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ World-writable files in home directory:${Q_COLORS[reset]}"
        echo "$world_writable" | while read -r file; do
            echo -e "    • $file"
        done
    fi

    # Check for suspicious hidden files
    local suspicious_hidden=$(find "$HOME" -name ".*" -type f -size +1M 2>/dev/null | head -10)
    if [[ -n "$suspicious_hidden" ]]; then
        echo -e "  ${Q_COLORS[warning]}⚠ Large hidden files found:${Q_COLORS[reset]}"
        echo "$suspicious_hidden" | while read -r file; do
            local size=$(du -h "$file" 2>/dev/null | cut -f1)
            echo -e "    • $(basename $file) ($size)"
        done
    fi

    # Check SSH key permissions
    if [[ -d "$HOME/.ssh" ]]; then
        local ssh_files=$(find "$HOME/.ssh" -type f -name "id_*" ! -name "*.pub")
        for key in $ssh_files; do
            local perm=$(stat -f "%p" "$key" 2>/dev/null || stat -c "%a" "$key")
            if [[ "$perm" != "600" ]] && [[ "$perm" != "400" ]]; then
                echo -e "  ${Q_COLORS[warning]}⚠ Incorrect SSH key permissions: $key ($perm)$
{Q_COLORS[reset]}"
            fi
        done
    fi

    # Check sudo access
    if groups $(whoami) | grep -q "admin\|sudo\|wheel"; then
        echo -e "  ${Q_COLORS[success]}✓ User has administrative privileges${Q_COLORS[reset]}"
    fi
}

quantum_map_dependencies() {
    echo -e "${Q_COLORS[info]}Mapping system dependencies...${Q_COLORS[reset]}"

    # Detect launch agents and daemons
    if [[ "$IS_MACOS" == "true" ]]; then
        local agents=$(ls ~/Library/LaunchAgents/*.plist 2>/dev/null | wc -l)
        echo -e "  ${Q_COLORS[info]}Launch Agents: $agents${Q_COLORS[reset]}"

        # List all launch agents
        for agent in ~/Library/LaunchAgents/*.plist 2>/dev/null; do
            local label=$(defaults read "${agent}" Label 2>/dev/null || echo "Unknown")
            echo -e "    • $(basename $agent) ($label)"
        done
    fi

    # Detect running services
    local services=$(launchctl list | grep -v "\-\|PID\|^$" | wc -l)
    echo -e "  ${Q_COLORS[info]}Running Services: $services${Q_COLORS[reset]}"

    # Detect open ports
    local open_ports=$(sudo lsof -i -nP 2>/dev/null | grep LISTEN | awk '{print $9}' | sort | uniq | wc -l)
    echo -e "  ${Q_COLORS[info]}Open Ports: $open_ports${Q_COLORS[reset]}"
}

# --- 3. QUANTUM DEEP CLEAN SYSTEM ---
quantum_deep_clean() {
    quantum_header
    echo -e "${Q_COLORS[grad1]}[🧹] QUANTUM DEEP CLEAN INITIATED${Q_COLORS[reset]}"
    echo

    # Create quantum snapshot before cleaning
    quantum_create_snapshot "pre-deep-clean"

    # Interactive confirmation
    echo -e "${Q_COLORS[warning]}⚠ WARNING: This will perform a deep clean of your system.$
{Q_COLORS[reset]}"
    echo -e "${Q_COLORS[warning]}   The following operations will be performed:${Q_COLORS[reset]}"
    echo -e "   1. Cache cleanup"
    echo -e "   2. Log file rotation"
    echo -e "   3. Orphan file removal"
    echo -e "   4. Broken symlink cleanup"
    echo -e "   5. Temporary file cleanup"
    echo -e "   6. System optimization"
    echo

    read -q "response?Proceed with deep clean? (y/N): "
    echo
    if [[ "$response" != "y" && "$response" != "Y" ]]; then
        echo -e "${Q_COLORS[error]}Deep clean cancelled.${Q_COLORS[reset]}"
        return 1
    fi

    # Execute cleaning operations
    echo
    quantum_clean_caches
    quantum_clean_logs
    quantum_clean_orphans
    quantum_clean_symlinks
    quantum_clean_temp
    quantum_optimize_system

    # Final report
    quantum_clean_report

    quantum_log "INFO" "Quantum deep clean completed"
}

quantum_create_snapshot() {
    local snapshot_name="$1"
    local snapshot_dir="${QNEXUS_SNAPSHOTS}/${snapshot_name}_$(date +%Y%m%d_%H%M%S)"

    mkdir -p "$snapshot_dir"
    echo -e "${Q_COLORS[info]}Creating system snapshot: $snapshot_name${Q_COLORS[reset]}"

    # Backup critical configuration files
    local config_files=(
        ~/.zshrc ~/.bashrc ~/.zprofile ~/.bash_profile
        ~/.gitconfig ~/.ssh/config ~/.ssh/known_hosts
    )

    for config in $config_files; do
        if [[ -f "$config" ]]; then
            cp "$config" "$snapshot_dir/" 2>/dev/null
        fi
    done

    # Backup application lists
    if command -v brew &> /dev/null; then
        brew list --formula > "$snapshot_dir/brew_formulae.txt" 2>/dev/null
        brew list --cask > "$snapshot_dir/brew_casks.txt" 2>/dev/null
    fi

    echo -e "  ${Q_COLORS[success]}✓ Snapshot created at: $snapshot_dir${Q_COLORS[reset]}"
    quantum_log "SNAPSHOT" "Created snapshot: $snapshot_name"
}

quantum_clean_caches() {
    echo -e "${Q_COLORS[grad2]}━━━━━━━ CLEANING CACHES ━━━━━━━${Q_COLORS[reset]}"

    local cache_dirs=(
        "$HOME/Library/Caches"
        "$HOME/.cache"
        "$HOME/Library/Developer/Xcode/DerivedData"
        "$HOME/Library/Containers/com.apple.Safari/Data/Library/Caches"
    )

    local total_freed=0

    for cache_dir in $cache_dirs; do
        if [[ -d "$cache_dir" ]]; then
            local before=$(du -sk "$cache_dir" 2>/dev/null | cut -f1 || echo 0)

            # Safe cleanup - remove files older than 30 days
            find "$cache_dir" -type f -atime +30 -delete 2>/dev/null
            find "$cache_dir" -type d -empty -delete 2>/dev/null

            local after=$(du -sk "$cache_dir" 2>/dev/null | cut -f1 || echo 0)
            local freed=$(( before - after ))

            if [[ $freed -gt 0 ]]; then
                echo -e "  ${Q_COLORS[success]}✓ Cleared $(basename $cache_dir): ${freed}KB$
{Q_COLORS[reset]}"
                (( total_freed += freed ))
            fi
        fi
    done

    # Clean specific application caches
    quantum_clean_app_caches
    echo -e "  ${Q_COLORS[info]}Total cache space freed: $((total_freed / 1024))MB${Q_COLORS[reset]}"
    quantum_log "CLEAN" "Cleared caches: ${total_freed}KB freed"
}

quantum_clean_app_caches() {
    # Application-specific cache cleaning
    local app_caches=(
        "com.apple.dt.Xcode"           # Xcode
        "com.google.Chrome"            # Chrome
        "com.apple.Safari"             # Safari
        "com.microsoft.VSCode"         # VS Code
        "com.spotify.client"           # Spotify
        "com.tinyspeck.slackmacgap"    # Slack
    )

    for app in $app_caches; do
        local cache_path="$HOME/Library/Caches/$app"
        if [[ -d "$cache_path" ]]; then
            local size=$(du -sk "$cache_path" 2>/dev/null | cut -f1)
            rm -rf "$cache_path" 2>/dev/null
            if [[ $? -eq 0 ]] && [[ $size -gt 0 ]]; then
                echo -e "    ${Q_COLORS[debug]}Cleared $app cache: ${size}KB${Q_COLORS[reset]}"
            fi
        fi
    done
}

quantum_clean_logs() {
    echo -e "${Q_COLORS[grad3]}──────── ROTATING LOG FILES ────────${Q_COLORS[reset]}"

    local log_dirs=(
        "$HOME/Library/Logs"
        "/var/log"
        "$HOME/.npm/_logs"
        "$HOME/Library/Application Support/Google/Chrome/Default/Logs"
    )

    local logs_cleared=0

    for log_dir in $log_dirs; do
        if [[ -d "$log_dir" ]]; then
            # Compress old logs
            find "$log_dir" -name "*.log" -type f -mtime +7 -exec gzip {} \; 2>/dev/null

            # Remove very old logs
            local cleared=$(find "$log_dir" -name "*.log.gz" -type f -mtime +30 -delete 2>/dev/null | wc -l)
            (( logs_cleared += cleared ))
        fi
    done

    echo -e "  ${Q_COLORS[success]}✓ Rotated $logs_cleared log files${Q_COLORS[reset]}"
    quantum_log "CLEAN" "Rotated $logs_cleared log files"
}

quantum_clean_orphans() {
    echo -e "${Q_COLORS[grad4]}──────── REMOVING ORPHANED FILES ────────${Q_COLORS[reset]}"
    # Define orphan patterns
    local orphan_patterns=(
        "*.tmp"
        "*.temp"
        "*.bak"
        "*.backup"
        "*.old"
        "*.swp"
        "*.swo"
        ".DS_Store"
        "Thumbs.db"
        "desktop.ini"
    )

    local orphans_removed=0

    # Search in common locations
    local search_paths=(
        "$HOME/Downloads"
        "$HOME/Desktop"
        "$HOME/Documents"
        "$HOME/.Trash"
        "/tmp"
    )

    for path in $search_paths; do
        if [[ -d "$path" ]]; then
            for pattern in $orphan_patterns; do
                local files=$(find "$path" -name "$pattern" -type f -atime +30 2>/dev/null)
                local count=$(echo "$files" | wc -l)

                if [[ $count -gt 0 ]]; then
                    echo "$files" | while read -r file; do
                        rm -f "$file" 2>/dev/null
                        (( orphans_removed++ ))
                    done
                fi
            done
        fi
    done

    echo -e "  ${Q_COLORS[success]}✓ Removed $orphans_removed orphaned files${Q_COLORS[reset]}"
    quantum_log "CLEAN" "Removed $orphans_removed orphaned files"
}

quantum_clean_symlinks() {
    echo -e "${Q_COLORS[grad5]}━━━━━━━ FIXING BROKEN SYMLINKS ━━━━━━━${Q_COLORS[reset]}"

    local broken_links=0
    local fixed_links=0

    # Find all broken symlinks
    find "$HOME" -type l ! -exec test -e {} \; -print 2>/dev/null | while read -r link; do
        (( broken_links++ ))

        # Try to find the original target
        local target=$(readlink "$link")
        if [[ -n "$target" ]]; then
            # Check if target exists elsewhere
            local found_target=$(find "$HOME" -name "$(basename "$target")" -type f 2>/dev/null | head -1)

            if [[ -n "$found_target" ]]; then
                # Fix the symlink
                ln -sf "$found_target" "$link" 2>/dev/null
                (( fixed_links++ ))
            else
                # Remove the broken symlink
                rm "$link" 2>/dev/null
            fi
        else
            rm "$link" 2>/dev/null
        fi
    done

    echo -e "  ${Q_COLORS[success]}✓ Fixed $fixed_links broken symlinks${Q_COLORS[reset]}"
    echo -e "  ${Q_COLORS[info]}Found $broken_links broken symlinks total${Q_COLORS[reset]}"
    quantum_log "CLEAN" "Fixed $fixed_links broken symlinks"
}

quantum_clean_temp() {
    echo -e "${Q_COLORS[grad6]}──────── CLEANING TEMPORARY FILES ────────$
{Q_COLORS[reset]}"

    local temp_dirs=(
        "/tmp"
        "/var/tmp"
        "$TMPDIR"
        "$HOME/tmp"
    )

    local temp_freed=0

    for temp_dir in $temp_dirs; do
        if [[ -d "$temp_dir" ]]; then
            local before=$(du -sk "$temp_dir" 2>/dev/null | cut -f1 || echo 0)

            # Remove files older than 1 day
            find "$temp_dir" -type f -atime +1 -delete 2>/dev/null
            find "$temp_dir" -type d -empty -atime +1 -delete 2>/dev/null

            local after=$(du -sk "$temp_dir" 2>/dev/null | cut -f1 || echo 0)
            local freed=$(( before - after ))

            if [[ $freed -gt 0 ]]; then
                echo -e "  ${Q_COLORS[success]}✓ Cleared $(basename $temp_dir): ${freed}KB$
{Q_COLORS[reset]}"
                (( temp_freed += freed ))
            fi
        fi
    done

    echo -e "  ${Q_COLORS[info]}Temporary space freed: $((temp_freed / 1024))MB${Q_COLORS[reset]}"
    quantum_log "CLEAN" "Cleared temp files: ${temp_freed}KB freed"
```

```
}

quantum_optimize_system() {
  echo -e "${Q_COLORS[grad1]}━━━━━━ SYSTEM OPTIMIZATION ━━━━━━${Q_COLORS[reset]}"

  # macOS-specific optimizations
  if [[ "$IS_MACOS" == "true" ]]; then
    echo -e "  ${Q_COLORS[info]}Optimizing macOS...${Q_COLORS[reset]}"

    # Clear DNS cache
    sudo dscacheutil -flushcache 2>/dev/null
    sudo killall -HUP mDNSResponder 2>/dev/null
    echo -e "    ${Q_COLORS[success]}✓ DNS cache cleared${Q_COLORS[reset]}"

    # Clear font cache
    sudo atsutil databases -removeUser 2>/dev/null
    sudo atsutil server -shutdown 2>/dev/null
    sudo atsutil server -ping 2>/dev/null
    echo -e "    ${Q_COLORS[success]}✓ Font cache cleared${Q_COLORS[reset]}"

    # Rebuild Spotlight index
    sudo mdutil -E / 2>/dev/null
    echo -e "    ${Q_COLORS[success]}✓ Spotlight index rebuilt${Q_COLORS[reset]}"

    # Clean mail attachments
    if [[ -d "$HOME/Library/Mail" ]]; then
      find "$HOME/Library/Mail" -name "*.emlxpart" -type f -delete 2>/dev/null
      echo -e "    ${Q_COLORS[success]}✓ Mail attachments cleaned${Q_COLORS[reset]}"
    fi
  fi

  # Optimize PATH
  quantum_optimize_path

  # Update package managers
  quantum_update_packages

  quantum_log "OPTIMIZE" "System optimization completed"
}

quantum_optimize_path() {
  echo -e "  ${Q_COLORS[info]}Optimizing PATH...${Q_COLORS[reset]}"

  # Remove duplicates from PATH
  local new_path=""
  local seen_paths=()
  local path_entries=(${(s.:.)PATH})

  for entry in $path_entries; do
    # Normalize path
    local normalized=$(cd "$entry" 2>/dev/null && pwd || echo "$entry")

    # Check if we've seen this path
    if [[ ! " ${seen_paths[@]} " =~ " ${normalized} " ]]; then
      seen_paths+=("$normalized")
      new_path="${new_path}:${normalized}"
    fi
  done

  # Remove leading colon
  new_path="${new_path#:}"

  # Set new PATH
  export PATH="$new_path"

  echo -e "    ${Q_COLORS[success]}✓ PATH optimized (${#seen_paths[@]} unique entries)$
{Q_COLORS[reset]}"
}

quantum_update_packages() {
  echo -e "  ${Q_COLORS[info]}Updating package managers...${Q_COLORS[reset]}"

  # Homebrew updates
  if command -v brew &> /dev/null; then
    brew update 2>/dev/null
    brew upgrade 2>/dev/null
    brew cleanup 2>/dev/null
    echo -e "    ${Q_COLORS[success]}✓ Homebrew updated${Q_COLORS[reset]}"
  fi

  # npm updates
  if command -v npm &> /dev/null; then
    npm update -g 2>/dev/null
    echo -e "    ${Q_COLORS[success]}✓ npm packages updated${Q_COLORS[reset]}"
  fi

  # pip updates
  if command -v pip3 &> /dev/null; then
    pip3 list --outdated --format=freeze 2>/dev/null | cut -d= -f1 | xargs -n1 pip3 install -U 2>/dev/null
    echo -e "    ${Q_COLORS[success]}✓ pip packages updated${Q_COLORS[reset]}"
  fi
}

quantum_clean_report() {
  echo
  echo -e "${Q_COLORS[grad2]}━━━━━━━ CLEANUP REPORT ━━━━━━━${Q_COLORS[reset]}"

  # Calculate disk space saved
  local disk_info=$(df -h / | tail -1)
  local available=$(echo $disk_info | awk '{print $4}')
  local capacity=$(echo $disk_info | awk '{print $5}')

  echo -e "  ${Q_COLORS[success]}✓ Deep clean completed successfully!${Q_COLORS[reset]}"
  echo -e "  ${Q_COLORS[info]}System Status:${Q_COLORS[reset]}"
  echo -e "    • Disk Available: $available"
  echo -e "    • Disk Usage: $capacity"

  # Show quantum health improvement
  quantum_show_health_improvement

  echo
  echo -e "${Q_COLORS[grad1]}[🎉] QUANTUM NEXUS DEEP CLEAN COMPLETE$
{Q_COLORS[reset]}"
  echo -e "${Q_COLORS[info]}Log saved to: $QNEXUS_LOG${Q_COLORS[reset]}"
```

```
}

quantum_show_health_improvement() {
    # Simulate health improvement
    local improvement=$((RANDOM % 20 + 10))
    echo -e "    • System Health Improvement: +${improvement}%"

    # Animated progress bar
    echo -n "    • Optimization Level: ["
    for i in {1..20}; do
        if [[ $i -le $((improvement / 5)) ]]; then
            echo -n "${Q_COLORS[success]}█"
        else
            echo -n "${Q_COLORS[debug]}░"
        fi
    done
    echo -e "${Q_COLORS[reset]}]"
}

# --- 4. QUANTUM MALWARE DETECTION SYSTEM ---
quantum_malware_scan() {
    quantum_header
    echo -e "${Q_COLORS[grad1]}[🛡️] QUANTUM MALWARE SCAN INITIATED${Q_COLORS[reset]}"
    echo

    # Known malware signatures (partial list for demonstration)
    local malware_signatures=(
        "cryptominer"
        "xmrig"
        "ccminer"
        "cpuminer"
       #!/usr/bin/env zsh
# ===============================================================================
# QUANTUM NEXUS DOTFILE MANAGEMENT MODULE v8.0
# ===============================================================================
# Complete Dotfile Analysis, Cleanup, Organization, and Migration
# ===============================================================================

# Load this module: source "${QNEXUS_MODULES}/dotfiles.zsh"

manage_dotfiles() {
    quantum_log "DOTFILES" "Starting military-grade dotfile management"

    echo
" ╔══════════════════════════════════════════════════════════════════════════
══════════════════════════════════════════╗ "
    echo " ║                    DOTFILE MANAGEMENT SYSTEM                      ║ "
    echo
" ╠══════════════════════════════════════════════════════════════════════════
══════════════════════════════════════════╣ "
    echo ""

    # 1. Comprehensive Dotfile Analysis
    echo "1. COMPREHENSIVE DOTFILE ANALYSIS"
    analyze_dotfiles
    # 2. Dotfile Cleanup & Removal
    echo ""
    echo "2. DOTFILE CLEANUP & REMOVAL"
    cleanup_dotfiles

    # 3. Dotfile Organization
    echo ""
    echo "3. DOTFILE ORGANIZATION"
    organize_dotfiles

    # 4. Dotfile Migration
    echo ""
    echo "4. DOTFILE MIGRATION"
    migrate_dotfiles

    # 5. Dotfile Backup & Versioning
    echo ""
    echo "5. DOTFILE BACKUP & VERSIONING"
    backup_dotfiles

    # 6. Dotfile Security Audit
    echo ""
    echo "6. DOTFILE SECURITY AUDIT"
    audit_dotfile_security

    quantum_log "DOTFILES" "Dotfile management completed"
}

analyze_dotfiles() {
    echo "  • Performing comprehensive dotfile analysis..."

    local dotfile_report="${QNEXUS_REPORTS}/dotfile_analysis_$(date +%Y%m%d_%H%M%S).json"
    local total_dotfiles=0
    local total_size=0

    # Find all dotfiles in home directory
    echo "    • Scanning for dotfiles..."
    local all_dotfiles=$(find "$HOME" -maxdepth 2 -name ".*" -type f 2>/dev/null | sort)

    cat > "$dotfile_report" << EOF
{
    "analysis_timestamp": "$(date -u -Iseconds)",
    "home_directory": "$HOME",
    "user": "$SYSTEM_USER",
    "dotfiles": [
EOF

    # Categorize dotfiles by type
    declare -A dotfile_categories=(
        ["SHELL"]=".zshrc .bashrc .bash_profile .zprofile .profile .zlogin .zlogout .zshenv"
        ["TERMINAL"]=".tmux.conf .screenrc .inputrc .editrc"
        ["EDITOR"]=".vimrc .viminfo .nvim .emacs .emacs.d .vscode .vscodium"
        ["GIT"]=".gitconfig .gitignore .gitattributes .git-credentials"
        ["SSH"]=".ssh .ssh/config .ssh/authorized_keys .ssh/known_hosts"
        ["AWS"]=".aws .aws/config .aws/credentials"
        ["DOCKER"]=".docker .docker/config.json"
        ["KUBERNETES"]=".kube .kube/config"
        ["NODE"]=".npmrc .npm .nvm .node_repl_history"
        ["PYTHON"]=".python_history .pythonrc .pyenv .pip"
        ["RUBY"]=".gemrc .ruby-version .ruby-gemset .rbenv"
        ["JAVA"]=".java .gradle .m2"
        ["GO"]=".go .gvm"
        ["RUST"]=".cargo .rustup"
        ["PHP"]=".php_history .composer"
        ["HASKELL"]=".ghc .ghci .stack"
        ["CONFIGURATION"]=".config .local .cache .gnupg .pki"
        ["APPLICATION"]=".docker .vscode .atom .sublime .iterm2"
        ["OTHER"]=""
    )

    # Process each dotfile
    while IFS= read -r dotfile; do
        ((total_dotfiles++))

        local filename=$(basename "$dotfile")
        local size=$(stat -f "%z" "$dotfile" 2>/dev/null || stat -c "%s" "$dotfile")
        local modified=$(stat -f "%Sm" -t "%Y-%m-%d %H:%M:%S" "$dotfile" 2>/dev/null || stat -c "%y"
"$dotfile")
        local owner=$(stat -f "%Su" "$dotfile" 2>/dev/null || stat -c "%U" "$dotfile")
        local permissions=$(stat -f "%Sp" "$dotfile" 2>/dev/null || stat -c "%A" "$dotfile")

        ((total_size += size))

        # Determine category
        local category="OTHER"
        for cat in "${(@k)dotfile_categories}"; do
            for pattern in ${=dotfile_categories[$cat]}; do
                if [[ "$filename" == "$pattern" ]] || [[ "$dotfile" =~ "$pattern" ]]; then
                    category="$cat"
                    break 2
                fi
            done
        done

        # Calculate dotfile age in days
        local current_time=$(date +%s)
        local file_mtime=$(stat -f "%m" "$dotfile" 2>/dev/null || stat -c "%Y" "$dotfile")
        local age_days=$(( (current_time - file_mtime) / 86400 ))

        cat >> "$dotfile_report" << EOF
        {
            "path": "$dotfile",
            "filename": "$filename",
            "size_bytes": $size,
            "size_human": "$(numfmt --to=iec-i --suffix=B $size 2>/dev/null || echo "${size}B")",
            "last_modified": "$modified",
            "age_days": $age_days,
            "owner": "$owner",
            "permissions": "$permissions",
            "category": "$category"
        },
EOF

        # Display progress for large analyses
        if [[ $total_dotfiles -lt 50 ]] || [[ $((total_dotfiles % 50)) -eq 0 ]]; then
            echo -n "."
        fi

    done <<< "$all_dotfiles"

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$dotfile_report" 2>/dev/null || sed -i '$ s/,$//' "$dotfile_report"

    # Calculate statistics by category
    cat >> "$dotfile_report" << EOF
    ],
    "statistics": {
        "total_dotfiles": $total_dotfiles,
        "total_size_bytes": $total_size,
        "total_size_human": "$(numfmt --to=iec-i --suffix=B $total_size 2>/dev/null || echo "${total_size}B")",
        "categories": {
EOF

    # Count by category
    for category in "${(@k)dotfile_categories}"; do
        local count=$(grep -c "\"category\": \"$category\"" "$dotfile_report")
        local category_size=$(grep -A 10 "\"category\": \"$category\"" "$dotfile_report" | grep "size_bytes" | awk
'{sum += $2} END {print sum}' | sed 's/,//')
        cat >> "$dotfile_report" << EOF
            "$category": {
                "count": $count,
                "size_bytes": ${category_size:-0}
            },
EOF
    done

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$dotfile_report" 2>/dev/null || sed -i '$ s/,$//' "$dotfile_report"

    cat >> "$dotfile_report" << EOF
        }
    }
}
EOF

    echo ""
    echo "    ✓ Dotfile analysis complete:"
    echo "      - Total dotfiles: $total_dotfiles"
    echo "      - Total size: $(numfmt --to=iec-i --suffix=B $total_size 2>/dev/null || echo "${total_size}B")"
    echo "      - Report: $dotfile_report"

    # Display summary
    echo ""
    echo "    • Top 10 largest dotfiles:"
    grep -E '"path":|"size_bytes":' "$dotfile_report" | \
        paste -d ' ' - - | \
        sed 's/"path": "\(.*\)", "size_bytes": \([0-9]*\)/\2 \1/' | \
        sort -rn | head -10 | while read -r line; do
            local size=$(echo $line | awk '{print $1}')
            local path=$(echo $line | awk '{$1=""; print $0}' | sed 's/^ //')
            echo "      • $(basename $path): $(numfmt --to=iec-i --suffix=B $size 2>/dev/null || echo "${size}B")"
        done

    quantum_log "ANALYSIS" "Dotfile analysis completed: $total_dotfiles files, $total_size bytes"
}

cleanup_dotfiles() {
    echo "  • Performing military-grade dotfile cleanup..."

    local cleanup_report="${QNEXUS_REPORTS}/dotfile_cleanup_$(date +%Y%m%d_%H%M%S).json"
    local backup_dir="${QNEXUS_BACKUP}/dotfiles_$(date +%Y%m%d_%H%M%S)"
    local files_removed=0
    local space_freed=0

    # Create backup directory
    mkdir -p "$backup_dir"

    cat > "$cleanup_report" << EOF
{
    "cleanup_timestamp": "$(date -u -Iseconds)",
    "backup_directory": "$backup_dir",
    "operations": [
EOF

    # Phase 1: Remove broken/duplicate dotfiles
    echo "    • Phase 1: Removing broken and duplicate dotfiles..."
    cleanup_broken_dotfiles "$cleanup_report" "$backup_dir"

    # Phase 2: Clean up old/unused dotfiles
    echo "    • Phase 2: Cleaning up old and unused dotfiles..."
    cleanup_old_dotfiles "$cleanup_report" "$backup_dir"

    # Phase 3: Remove temporary dotfiles
    echo "    • Phase 3: Removing temporary dotfiles..."
    cleanup_temporary_dotfiles "$cleanup_report" "$backup_dir"

    # Phase 4: Clean application-specific dotfiles
    echo "    • Phase 4: Cleaning application dotfiles..."
    cleanup_application_dotfiles "$cleanup_report" "$backup_dir"

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$cleanup_report" 2>/dev/null || sed -i '$ s/,$//' "$cleanup_report"

    # Calculate totals
    files_removed=$(grep -c '"operation": "REMOVE"' "$cleanup_report")
    space_freed=$(grep '"size_bytes":' "$cleanup_report" | awk '{sum += $2} END {print sum}' | sed 's/,//')

    cat >> "$cleanup_report" << EOF
    ],
    "summary": {
        "files_removed": $files_removed,
        "space_freed_bytes": ${space_freed:-0},
        "space_freed_human": "$(numfmt --to=iec-i --suffix=B ${space_freed:-0} 2>/dev/null || echo "0B")",
        "backup_location": "$backup_dir"
    }
}
EOF
```

```
  echo ""
  echo "    ✓ Dotfile cleanup complete:"
  echo "      - Files removed: $files_removed"
  echo "      - Space freed: $(numfmt --to=iec-i --suffix=B ${space_freed:-0} 2>/dev/null || echo "0B")"
  echo "      - Backup: $backup_dir"
  echo "      - Report: $cleanup_report"

  quantum_log "CLEANUP" "Dotfile cleanup completed: $files_removed files removed, $space_freed bytes
freed"
}

cleanup_broken_dotfiles() {
  local report_file="$1"
  local backup_dir="$2"

  # Find broken symlinks in home directory
  find "$HOME" -maxdepth 3 -name ".*" -type l ! -exec test -e {} \; 2>/dev/null | while read -r broken_link; do
    local link_target=$(readlink "$broken_link")
    local size=$(stat -f "%z" "$broken_link" 2>/dev/null || stat -c "%s" "$broken_link" 2>/dev/null || echo 0)

    # Backup before removal
    cp -r "$broken_link" "$backup_dir/" 2>/dev/null

    # Remove the broken symlink
    rm -f "$broken_link"

    cat >> "$report_file" << EOF
    {
      "operation": "REMOVE",
      "type": "BROKEN_SYMLINK",
      "path": "$broken_link",
      "target": "$link_target",
      "size_bytes": $size,
      "timestamp": "$(date -u -Iseconds)"
    },
EOF
    echo "      • Removed broken symlink: $(basename $broken_link) → $link_target"
  done

  # Find duplicate dotfiles (same name in multiple locations)
  find "$HOME" -maxdepth 2 -name ".*" -type f 2>/dev/null | \
    xargs -I {} basename {} | \
    sort | uniq -d | while read -r duplicate; do
      local duplicates=($(find "$HOME" -maxdepth 2 -name "$duplicate" -type f 2>/dev/null))

      # Keep the newest, backup and remove others
      if [[ ${#duplicates[@]} -gt 1 ]]; then
        # Find newest file
        local newest=""
        local newest_time=0

        for dup in "${duplicates[@]}"; do
          local mtime=$(stat -f "%m" "$dup" 2>/dev/null || stat -c "%Y" "$dup")
          if [[ $mtime -gt $newest_time ]]; then
            newest_time=$mtime
            newest="$dup"
          fi
```

```
            done

            # Backup and remove older duplicates
            for dup in "${duplicates[@]}"; do
                if [[ "$dup" != "$newest" ]]; then
                    local size=$(stat -f "%z" "$dup" 2>/dev/null || stat -c "%s" "$dup")
                    cp -r "$dup" "$backup_dir/" 2>/dev/null
                    rm -f "$dup"

                    cat >> "$report_file" << EOF
    {
        "operation": "REMOVE",
        "type": "DUPLICATE",
        "path": "$dup",
        "kept_version": "$newest",
        "size_bytes": $size,
        "timestamp": "$(date -u -Iseconds)"
    },
EOF
                    echo "      • Removed duplicate: $(basename $dup) (kept: $(basename $newest))"
                fi
            done
        fi
    done
}

cleanup_old_dotfiles() {
    local report_file="$1"
    local backup_dir="$2"

    # Find dotfiles older than 1 year
    find "$HOME" -maxdepth 2 -name ".*" -type f -atime +365 2>/dev/null | while read -r old_file; do
        local filename=$(basename "$old_file")
        local size=$(stat -f "%z" "$old_file" 2>/dev/null || stat -c "%s" "$old_file")
        local last_access=$(stat -f "%Sa" -t "%Y-%m-%d" "$old_file" 2>/dev/null || stat -c "%x" "$old_file")

        # Check if file is likely unused
        local is_essential=false

        # Essential dotfiles to keep
        local essential_patterns=(
            ".zshrc" ".bashrc" ".bash_profile" ".profile"
            ".gitconfig" ".ssh/config" ".ssh/authorized_keys"
            ".aws/config" ".aws/credentials" ".kube/config"
        )

        for pattern in "${essential_patterns[@]}"; do
            if [[ "$filename" == "$pattern" ]] || [[ "$old_file" =~ "$pattern" ]]; then
                is_essential=true
                break
            fi
        done

        if [[ "$is_essential" == false ]]; then
            # Backup and remove
            cp -r "$old_file" "$backup_dir/" 2>/dev/null
            rm -f "$old_file"
            cat >> "$report_file" << EOF
        {
            "operation": "REMOVE",
            "type": "OLD_FILE",
            "path": "$old_file",
            "last_accessed": "$last_access",
            "size_bytes": $size,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      • Removed old dotfile: $filename (last access: $last_access)"
        fi
    done

    # Clean empty dot directories
    find "$HOME" -maxdepth 2 -name ".*" -type d -empty 2>/dev/null | while read -r empty_dir; do
        # Skip important empty directories
        if [[ "$empty_dir" != "$HOME/.ssh" ]] && [[ "$empty_dir" != "$HOME/.config" ]]; then
            cp -r "$empty_dir" "$backup_dir/" 2>/dev/null
            rmdir "$empty_dir" 2>/dev/null

            cat >> "$report_file" << EOF
        {
            "operation": "REMOVE",
            "type": "EMPTY_DIRECTORY",
            "path": "$empty_dir",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      • Removed empty directory: $(basename $empty_dir)"
        fi
    done
}

cleanup_temporary_dotfiles() {
    local report_file="$1"
    local backup_dir="$2"

    # Temporary file patterns
    local temp_patterns=(
        ".*.swp" ".*.swo" ".*.swn"  # Vim swap files
        ".*.tmp" ".*.temp"          # Temporary files
        ".*.bak" ".*.backup"        # Backup files
        ".*~" ".#*"                 # Backup and lock files
        ".DS_Store" "._.DS_Store"   # macOS metadata
        ".Trash" ".Trashes"         # Trash directories
        ".fseventsd" ".Spotlight*"  # macOS system files
        ".TemporaryItems"           # Temporary items
    )

    for pattern in "${temp_patterns[@]}"; do
        find "$HOME" -maxdepth 3 -name "$pattern" 2>/dev/null | while read -r temp_file; do
            local size=0
            if [[ -f "$temp_file" ]]; then
                size=$(stat -f "%z" "$temp_file" 2>/dev/null || stat -c "%s" "$temp_file")
                cp -r "$temp_file" "$backup_dir/" 2>/dev/null
                rm -f "$temp_file"
            elif [[ -d "$temp_file" ]]; then
                cp -r "$temp_file" "$backup_dir/" 2>/dev/null
                rm -rf "$temp_file"
            fi

            cat >> "$report_file" << EOF
        {
            "operation": "REMOVE",
            "type": "TEMPORARY_FILE",
            "path": "$temp_file",
            "pattern": "$pattern",
            "size_bytes": $size,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      • Removed temporary: $pattern"
        done
    done

    # Clean npm cache
    if [[ -d "$HOME/.npm/_logs" ]]; then
        local npm_logs_size=$(du -sk "$HOME/.npm/_logs" 2>/dev/null | cut -f1)
        cp -r "$HOME/.npm/_logs" "$backup_dir/" 2>/dev/null
        rm -rf "$HOME/.npm/_logs"

        cat >> "$report_file" << EOF
        {
            "operation": "REMOVE",
            "type": "NPM_LOGS",
            "path": "$HOME/.npm/_logs",
            "size_bytes": $((npm_logs_size * 1024)),
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
        echo "      • Removed npm logs: $((npm_logs_size / 1024)) MB"
    fi

    # Clean pip cache
    if [[ -d "$HOME/.cache/pip" ]]; then
        local pip_cache_size=$(du -sk "$HOME/.cache/pip" 2>/dev/null | cut -f1)
        cp -r "$HOME/.cache/pip" "$backup_dir/" 2>/dev/null
        rm -rf "$HOME/.cache/pip"

        cat >> "$report_file" << EOF
        {
            "operation": "REMOVE",
            "type": "PIP_CACHE",
            "path": "$HOME/.cache/pip",
            "size_bytes": $((pip_cache_size * 1024)),
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
        echo "      • Removed pip cache: $((pip_cache_size / 1024)) MB"
    fi
}
```

```
cleanup_application_dotfiles() {
    local report_file="$1"
    local backup_dir="$2"

    # Application-specific cleanup patterns
    declare -A app_cleanup=(
        ["CHROME_CACHE"]="$HOME/Library/Application Support/Google/Chrome/Default/Cache"
        ["SAFARI_CACHE"]="$HOME/Library/Caches/com.apple.Safari"
        ["FIREFOX_CACHE"]="$HOME/Library/Caches/Firefox"
        ["VSCODE_CACHE"]="$HOME/.vscode"
        ["ATOM_CACHE"]="$HOME/.atom/compile-cache"
        ["SLACK_CACHE"]="$HOME/Library/Caches/com.tinyspeck.slackmacgap"
        ["DOCKER_CACHE"]="$HOME/Library/Containers/com.docker.docker/Data"
        ["ZOOM_CACHE"]="$HOME/.zoom"
        ["TEAMS_CACHE"]="$HOME/Library/Caches/com.microsoft.teams"
    )

    for app_name in "${(@k)app_cleanup}"; do
        local app_path="${app_cleanup[$app_name]}"

        if [[ -d "$app_path" ]]; then
            local app_size=$(du -sk "$app_path" 2>/dev/null | cut -f1)

            # Create backup
            mkdir -p "$backup_dir/app_cache"
            cp -r "$app_path" "$backup_dir/app_cache/" 2>/dev/null

            # Clean cache files older than 7 days
            find "$app_path" -type f -atime +7 -delete 2>/dev/null

            local new_size=$(du -sk "$app_path" 2>/dev/null | cut -f1)
            local freed=$(( (app_size - new_size) * 1024 ))

            cat >> "$report_file" << EOF
        {
            "operation": "CLEAN",
            "type": "APP_CACHE",
            "application": "$app_name",
            "path": "$app_path",
            "original_size_bytes": $((app_size * 1024)),
            "new_size_bytes": $((new_size * 1024)),
            "freed_bytes": $freed,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      • Cleaned $app_name cache: $((freed / 1024 / 1024)) MB freed"
        fi
    done
}

organize_dotfiles() {
    echo "  • Organizing dotfiles into structured layout..."

    local organize_report="${QNEXUS_REPORTS}/dotfile_organization_$(date +
%Y%m%d_%H%M%S).json"
    local organized_base="${HOME}/.config/dotfiles_organized"
    # Create organized directory structure
    local organized_dirs=(
        "${organized_base}/shell"
        "${organized_base}/git"
        "${organized_base}/ssh"
        "${organized_base}/aws"
        "${organized_base}/kubernetes"
        "${organized_base}/docker"
        "${organized_base}/editors"
        "${organized_base}/terminals"
        "${organized_base}/languages"
        "${organized_base}/applications"
        "${organized_base}/misc"
    )

    for dir in "${organized_dirs[@]}"; do
        mkdir -p "$dir"
    done

    cat > "$organize_report" << EOF
{
    "organization_timestamp": "$(date -u -Iseconds)",
    "organized_base": "$organized_base",
    "operations": [
EOF

    # Organization mapping: source -> destination
    declare -A org_mapping=(
        # Shell configurations
        ["$HOME/.zshrc"]="${organized_base}/shell/zshrc"
        ["$HOME/.bashrc"]="${organized_base}/shell/bashrc"
        ["$HOME/.bash_profile"]="${organized_base}/shell/bash_profile"
        ["$HOME/.profile"]="${organized_base}/shell/profile"
        ["$HOME/.zprofile"]="${organized_base}/shell/zprofile"

        # Git configurations
        ["$HOME/.gitconfig"]="${organized_base}/git/config"
        ["$HOME/.gitignore"]="${organized_base}/git/ignore"
        ["$HOME/.gitattributes"]="${organized_base}/git/attributes"

        # SSH configurations
        ["$HOME/.ssh/config"]="${organized_base}/ssh/config"
        ["$HOME/.ssh/known_hosts"]="${organized_base}/ssh/known_hosts"

        # AWS configurations
        ["$HOME/.aws/config"]="${organized_base}/aws/config"
        ["$HOME/.aws/credentials"]="${organized_base}/aws/credentials"

        # Kubernetes configurations
        ["$HOME/.kube/config"]="${organized_base}/kubernetes/config"

        # Editor configurations
        ["$HOME/.vimrc"]="${organized_base}/editors/vimrc"
        ["$HOME/.vim"]="${organized_base}/editors/vim"
        ["$HOME/.config/nvim"]="${organized_base}/editors/nvim"

        # Terminal configurations
        ["$HOME/.tmux.conf"]="${organized_base}/terminals/tmux.conf"
        ["$HOME/.config/alacritty"]="${organized_base}/terminals/alacritty"
        ["$HOME/.config/kitty"]="${organized_base}/terminals/kitty"
    )

    local files_organized=0

    # Move and symlink dotfiles
    for source in "${(@k)org_mapping}"; do
        local dest="${org_mapping[$source]}"

        if [[ -e "$source" ]]; then
            local filename=$(basename "$source")
            local size=$(stat -f "%z" "$source" 2>/dev/null || stat -c "%s" "$source" 2>/dev/null || echo 0)

            # Create parent directory if needed
            mkdir -p "$(dirname "$dest")"

            # Move the file
            mv "$source" "$dest" 2>/dev/null

            # Create symlink back to original location
            ln -sf "$dest" "$source"

            cat >> "$organize_report" << EOF
        {
            "operation": "ORGANIZE",
            "source": "$source",
            "destination": "$dest",
            "size_bytes": $size,
            "symlink_created": true,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            ((files_organized++))
            echo "      • Organized: $filename → $(basename $(dirname "$dest"))/"
        fi
    done

    # Organize .config directory
    echo "    • Organizing .config directory..."
    if [[ -d "$HOME/.config" ]]; then
        local config_items=$(find "$HOME/.config" -maxdepth 1 -type d | wc -l)

        # Move large config directories to organized location
        find "$HOME/.config" -maxdepth 1 -type d -size +1M 2>/dev/null | while read -r config_dir; do
            local dirname=$(basename "$config_dir")
            if [[ "$dirname" != "." ]] && [[ "$dirname" != ".." ]] && [[ "$dirname" != "dotfiles_organized" ]]; then
                local size=$(du -sk "$config_dir" 2>/dev/null | cut -f1)

                # Move to organized location
                mv "$config_dir" "${organized_base}/applications/${dirname}" 2>/dev/null

                # Create symlink
                ln -sf "${organized_base}/applications/${dirname}" "$config_dir"

                cat >> "$organize_report" << EOF
        {
            "operation": "ORGANIZE_CONFIG",
            "directory": "$config_dir",
            "organized_location": "${organized_base}/applications/${dirname}",
            "size_kb": $size,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
                ((files_organized++))
                echo "      • Moved config: $dirname ($((size / 1024)) MB)"
            fi
        done
    fi

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$organize_report" 2>/dev/null || sed -i '$ s/,$//' "$organize_report"

    cat >> "$organize_report" << EOF
    ],
    "summary": {
        "files_organized": $files_organized,
        "organized_base": "$organized_base",
        "structure": {
            "shell": "$(ls -1 "${organized_base}/shell" 2>/dev/null | wc -l)",
            "git": "$(ls -1 "${organized_base}/git" 2>/dev/null | wc -l)",
            "ssh": "$(ls -1 "${organized_base}/ssh" 2>/dev/null | wc -l)",
            "aws": "$(ls -1 "${organized_base}/aws" 2>/dev/null | wc -l)",
            "applications": "$(ls -1 "${organized_base}/applications" 2>/dev/null | wc -l)"
        }
    }
}
EOF

    echo ""
    echo "    ✓ Dotfile organization complete:"
    echo "      - Files organized: $files_organized"
    echo "      - Organized base: $organized_base"
    echo "      - Report: $organize_report"

    # Create organization manifest
    cat > "${organized_base}/MANIFEST.md" << EOF
# Dotfile Organization Manifest

Created: $(date)
Organized by: Quantum Nexus v8.0
Base Directory: $organized_base

## Directory Structure

$(find "$organized_base" -type d | sed 's|.*\.config/dotfiles_organized/||' | grep -v '^$' | sort | sed 's/^/  /')

## Symlink Mapping

$(for source in "${(@k)org_mapping}"; do
    echo "- $source → ${org_mapping[$source]}"
done)
```

```
## Restoration Notes

To restore original structure, run:
\`\`\`bash
# Remove all symlinks
find $organized_base -type l -exec rm {} \\;

# Move files back to original locations
for file in $organized_base/**/*; do
    if [[ -f \$file ]]; then
        original_path="\$(echo \$file | sed 's|$organized_base/||')"
        mv "\$file" "\$HOME/.config/\$original_path"
    fi
done
\`\`\`
EOF

    quantum_log "ORGANIZATION" "Dotfile organization completed: $files_organized files organized"
}

migrate_dotfiles() {
    echo "  • Migrating dotfiles to new structure..."

    local migration_report="${QNEXUS_REPORTS}/dotfile_migration_$(date +%Y%m%d_%H%M%S).json"
    local migration_target=""

    # Determine migration target
    if [[ -d "/Volumes/Backup" ]]; then
        migration_target="/Volumes/Backup/dotfiles_$(date +%Y%m%d)"
    elif [[ -d "$HOME/Backups" ]]; then
        migration_target="$HOME/Backups/dotfiles_$(date +%Y%m%d)"
    else
        migration_target="$HOME/dotfiles_backup_$(date +%Y%m%d)"
    fi

    mkdir -p "$migration_target"

    cat > "$migration_report" << EOF
{
    "migration_timestamp": "$(date -u -Iseconds)",
    "source": "$HOME",
    "target": "$migration_target",
    "operations": [
EOF

    # Critical dotfiles to migrate
    local critical_dotfiles=(
        ".zshrc" ".bashrc" ".bash_profile" ".profile"
        ".gitconfig" ".gitignore" ".gitattributes"
        ".ssh/config" ".ssh/authorized_keys" ".ssh/known_hosts"
        ".aws/config" ".aws/credentials"
        ".kube/config"
        ".docker/config.json"
        ".vimrc" ".vim"
        ".tmux.conf"
        ".npmrc" ".pythonrc"
    )
    local files_migrated=0
    local total_size=0

    # Migrate critical dotfiles
    for dotfile in "${critical_dotfiles[@]}"; do
        local source_path="$HOME/$dotfile"

        if [[ -e "$source_path" ]]; then
            local size=$(stat -f "%z" "$source_path" 2>/dev/null || stat -c "%s" "$source_path" 2>/dev/null || echo
0)

            # Create target directory structure
            local target_dir="$migration_target/$(dirname "$dotfile")"
            mkdir -p "$target_dir"

            # Copy with preservation of attributes
            cp -a "$source_path" "$target_dir/"

            ((total_size += size))
            ((files_migrated++))

            cat >> "$migration_report" << EOF
        {
            "operation": "MIGRATE",
            "file": "$dotfile",
            "source": "$source_path",
            "target": "$target_dir/$dotfile",
            "size_bytes": $size,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      • Migrated: $dotfile"
        fi
    done

    # Migrate .config directory
    if [[ -d "$HOME/.config" ]]; then
        local config_size=$(du -sk "$HOME/.config" 2>/dev/null | cut -f1)

        # Copy .config directory
        cp -a "$HOME/.config" "$migration_target/"

        ((total_size += config_size * 1024))

        cat >> "$migration_report" << EOF
        {
            "operation": "MIGRATE_CONFIG",
            "directory": "$HOME/.config",
            "target": "$migration_target/.config",
            "size_bytes": $((config_size * 1024)),
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
        echo "      • Migrated .config directory: $((config_size / 1024)) MB"
    fi
    # Create migration manifest
    cat > "$migration_target/MANIFEST.txt" << EOF
Dotfile Migration Manifest
==========================
Date: $(date)
Source: $HOME
Target: $migration_target
Files Migrated: $files_migrated
Total Size: $(numfmt --to=iec-i --suffix=B $total_size 2>/dev/null || echo "${total_size}B")

Critical Files:
$(for dotfile in "${critical_dotfiles[@]}"; do
    if [[ -e "$HOME/$dotfile" ]]; then
        echo "- $dotfile"
    fi
done)

Restoration Command:
To restore dotfiles, run:
\`\`\`bash
cd "$migration_target"
find . -type f -exec cp -a {} "$HOME/" \\;
\`\`\`

Verification Command:
To verify migration, run:
\`\`\`bash
diff -r "$HOME" "$migration_target" | grep -v "Only in"
\`\`\`
EOF

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$migration_report" 2>/dev/null || sed -i '$ s/,$//' "$migration_report"

    cat >> "$migration_report" << EOF
    ],
    "summary": {
        "files_migrated": $files_migrated,
        "total_size_bytes": $total_size,
        "total_size_human": "$(numfmt --to=iec-i --suffix=B $total_size 2>/dev/null || echo "${total_size}B")",
        "migration_target": "$migration_target",
        "verification_checksum": "$(shasum -a 256 "$migration_target/MANIFEST.txt" 2>/dev/null | cut -d' ' -f1)"
    }
}
EOF

    echo ""
    echo "    ✓ Dotfile migration complete:"
    echo "      - Files migrated: $files_migrated"
    echo "      - Total size: $(numfmt --to=iec-i --suffix=B $total_size 2>/dev/null || echo "${total_size}B")"
    echo "      - Target location: $migration_target"
    echo "      - Manifest: $migration_target/MANIFEST.txt"
    echo "      - Report: $migration_report"

    quantum_log "MIGRATION" "Dotfile migration completed: $files_migrated files, $total_size bytes"
}
```

```
backup_dotfiles() {
    echo "  • Creating comprehensive dotfile backups..."

    local backup_report="${QNEXUS_REPORTS}/dotfile_backup_$(date +%Y%m%d_%H%M%S).json"
    local backup_dir="${QNEXUS_BACKUP}/dotfiles_complete_$(date +%Y%m%d_%H%M%S)"
    local encrypted_backup_dir="${backup_dir}_encrypted"

    mkdir -p "$backup_dir" "$encrypted_backup_dir"

    cat > "$backup_report" << EOF
{
    "backup_timestamp": "$(date -u -Iseconds)",
    "backup_directory": "$backup_dir",
    "encrypted_directory": "$encrypted_backup_dir",
    "operations": [
EOF

    # Phase 1: Full dotfile backup
    echo "    • Phase 1: Creating full dotfile backup..."
    backup_full_dotfiles "$backup_report" "$backup_dir"

    # Phase 2: Versioned backup
    echo "    • Phase 2: Creating versioned backup..."
    create_versioned_backup "$backup_report" "$backup_dir"

    # Phase 3: Encrypted backup
    echo "    • Phase 3: Creating encrypted backup..."
    create_encrypted_backup "$backup_report" "$backup_dir" "$encrypted_backup_dir"

    # Phase 4: Cloud backup
    echo "    • Phase 4: Creating cloud backup..."
    create_cloud_backup "$backup_report" "$backup_dir"

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$backup_report" 2>/dev/null || sed -i '$ s/,$//' "$backup_report"

    # Calculate backup statistics
    local backup_size=$(du -sk "$backup_dir" 2>/dev/null | cut -f1)
    local file_count=$(find "$backup_dir" -type f | wc -l)

    cat >> "$backup_report" << EOF
    ],
    "summary": {
        "backup_size_bytes": $((backup_size * 1024)),
        "backup_size_human": "$(numfmt --to=iec-i --suffix=B $((backup_size * 1024)) 2>/dev/null)",
        "file_count": $file_count,
        "backup_location": "$backup_dir",
        "encrypted_backup": "$encrypted_backup_dir",
        "integrity_check": "$(shasum -a 256 "$backup_dir"/*.tar.gz 2>/dev/null | head -1 | cut -d' ' -f1 || echo "N/
A")"
    }
}
EOF

    echo ""
    echo "    ✓ Dotfile backup complete:"
    echo "      - Backup size: $((backup_size / 1024)) MB"
    echo "      - Files backed up: $file_count"
    echo "      - Backup location: $backup_dir"
    echo "      - Encrypted backup: $encrypted_backup_dir"
    echo "      - Report: $backup_report"

    # Create restoration script
    cat > "$backup_dir/RESTORE.sh" << 'EOF'
#!/bin/bash
# Dotfile Restoration Script
# Created: $(date)

BACKUP_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
HOME_DIR="$HOME"
LOG_FILE="${BACKUP_DIR}/restoration_$(date +%Y%m%d_%H%M%S).log"

log_message() {
    echo "[$(date)] $1" >> "$LOG_FILE"
    echo "$1"
}

restore_dotfiles() {
    log_message "Starting dotfile restoration..."

    # Extract compressed backups
    for archive in "$BACKUP_DIR"/*.tar.gz; do
        if [[ -f "$archive" ]]; then
            log_message "Extracting: $(basename $archive)"
            tar -xzf "$archive" -C "$HOME_DIR" 2>> "$LOG_FILE"
        fi
    done

    # Restore from organized structure
    if [[ -d "$BACKUP_DIR/organized" ]]; then
        log_message "Restoring organized structure..."
        cp -r "$BACKUP_DIR/organized"/* "$HOME_DIR/.config/" 2>> "$LOG_FILE"
    fi

    # Restore encrypted backups if key is provided
    if [[ -f "$BACKUP_DIR/backup.key" ]] && [[ -d "$BACKUP_DIR/encrypted" ]]; then
        read -p "Encrypted backup detected. Restore? (y/N): " restore_encrypted
        if [[ "$restore_encrypted" == "y" ]]; then
            log_message "Restoring encrypted backups..."
            # Decryption logic would go here
        fi
    fi

    log_message "Restoration complete"
    log_message "Log saved to: $LOG_FILE"
}

# Safety check
read -p "Restore dotfiles from backup? This will overwrite existing files. (y/N): " confirm
if [[ "$confirm" == "y" || "$confirm" == "Y" ]]; then
    restore_dotfiles
else
    echo "Restoration cancelled"
    exit 1
```

```
fi
EOF

    chmod +x "$backup_dir/RESTORE.sh"

    quantum_log "BACKUP" "Dotfile backup completed: $((backup_size / 1024)) MB, $file_count files"
}

backup_full_dotfiles() {
    local report_file="$1"
    local backup_dir="$2"

    # Create timestamped backup
    local timestamp=$(date +%Y%m%d_%H%M%S)
    local backup_file="${backup_dir}/dotfiles_full_${timestamp}.tar.gz"

    # Find and backup all dotfiles
    find "$HOME" -maxdepth 2 -name ".*" -type f 2>/dev/null | \
        tar -czf "$backup_file" -T - 2>/dev/null

    local backup_size=$(stat -f "%z" "$backup_file" 2>/dev/null || stat -c "%s" "$backup_file")
    local file_count=$(tar -tzf "$backup_file" 2>/dev/null | wc -l)

    cat >> "$report_file" << EOF
        {
            "operation": "FULL_BACKUP",
            "backup_file": "$backup_file",
            "size_bytes": $backup_size,
            "file_count": $file_count,
            "compression": "gzip",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
    echo "      • Full backup: $(basename $backup_file) ($((backup_size / 1024 / 1024)) MB, $file_count files)"

    # Create incremental backup of changed files
    local incremental_file="${backup_dir}/dotfiles_incremental_${timestamp}.tar.gz"
    find "$HOME" -maxdepth 2 -name ".*" -type f -mtime -1 2>/dev/null | \
        tar -czf "$incremental_file" -T - 2>/dev/null

    local incremental_size=$(stat -f "%z" "$incremental_file" 2>/dev/null || stat -c "%s" "$incremental_file")
    local incremental_count=$(tar -tzf "$incremental_file" 2>/dev/null | wc -l)

    cat >> "$report_file" << EOF
        {
            "operation": "INCREMENTAL_BACKUP",
            "backup_file": "$incremental_file",
            "size_bytes": $incremental_size,
            "file_count": $incremental_count,
            "time_frame": "24_hours",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
    echo "      • Incremental backup: $(basename $incremental_file) ($((incremental_size / 1024)) KB,
$incremental_count files)"
}
```

```
create_versioned_backup() {
    local report_file="$1"
    local backup_dir="$2"

    # Create versioned backup structure
    local versioned_dir="${backup_dir}/versioned"
    mkdir -p "$versioned_dir"

    # Backup with git-like versioning
    local current_version="v1.0.0"
    local version_file="${versioned_dir}/VERSION"

    echo "$current_version" > "$version_file"

    # Create versioned backup of critical dotfiles
    local critical_files=(
        ".zshrc" ".bashrc" ".gitconfig" ".ssh/config"
        ".aws/config" ".kube/config" ".vimrc"
    )

    for file in "${critical_files[@]}"; do
        local source_file="$HOME/$file"
        if [[ -f "$source_file" ]]; then
            local versioned_file="${versioned_dir}/${file//\//_}_${current_version}"
            cp -a "$source_file" "$versioned_file"

            # Calculate diff from previous version if exists
            local previous_version=$(ls "${versioned_dir}/${file//\//_}_v*" 2>/dev/null | grep -v "$current_version" |
tail -1)
            if [[ -f "$previous_version" ]]; then
                local diff_output=$(diff -u "$previous_version" "$versioned_file" 2>/dev/null || true)
                echo "$diff_output" > "${versioned_file}.diff"
            fi
        fi
    done

    local versioned_count=$(find "$versioned_dir" -type f | wc -l)

    cat >> "$report_file" << EOF
        {
            "operation": "VERSIONED_BACKUP",
            "directory": "$versioned_dir",
            "version": "$current_version",
            "file_count": $versioned_count,
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
    echo "     • Versioned backup: $current_version ($versioned_count files)"
}

create_encrypted_backup() {
    local report_file="$1"
    local backup_dir="$2"
    local encrypted_dir="$3"

    # Check for encryption tools
    if command -v gpg &>/dev/null; then
```

```
        echo "      • Creating encrypted backup with GPG..."

        # Create encryption key
        local encryption_key="${backup_dir}/backup.key"
        openssl rand -base64 32 > "$encryption_key" 2>/dev/null

        # Encrypt the backup
        for backup_file in "$backup_dir"/*.tar.gz; do
            if [[ -f "$backup_file" ]]; then
                local encrypted_file="${encrypted_dir}/$(basename $backup_file).gpg"
                gpg --batch --yes --passphrase-file "$encryption_key" \
                    --symmetric --cipher-algo AES256 \
                    --output "$encrypted_file" "$backup_file" 2>/dev/null

                if [[ -f "$encrypted_file" ]]; then
                    local encrypted_size=$(stat -f "%z" "$encrypted_file" 2>/dev/null || stat -c "%s"
"$encrypted_file")

                    cat >> "$report_file" << EOF
        {
            "operation": "ENCRYPTED_BACKUP",
            "original": "$(basename $backup_file)",
            "encrypted": "$(basename $encrypted_file)",
            "size_bytes": $encrypted_size,
            "encryption": "AES256_GPG",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
                    echo "        • Encrypted: $(basename $backup_file) → $(basename $encrypted_file)"
                fi
            fi
        done

        # Secure the encryption key
        chmod 400 "$encryption_key"
        echo "      ✓ Encryption key: $encryption_key (keep secure!)"

    elif command -v openssl &>/dev/null; then
        echo "      • Creating encrypted backup with OpenSSL..."

        # Create encryption key
        local encryption_key="${backup_dir}/backup.key"
        openssl rand -base64 32 > "$encryption_key" 2>/dev/null

        # Encrypt with OpenSSL
        for backup_file in "$backup_dir"/*.tar.gz; do
            if [[ -f "$backup_file" ]]; then
                local encrypted_file="${encrypted_dir}/$(basename $backup_file).enc"
                openssl enc -aes-256-cbc -salt -in "$backup_file" \
                    -out "$encrypted_file" -pass file:"$encryption_key" 2>/dev/null

                if [[ -f "$encrypted_file" ]]; then
                    local encrypted_size=$(stat -f "%z" "$encrypted_file" 2>/dev/null || stat -c "%s"
"$encrypted_file")

                    cat >> "$report_file" << EOF
        {
            "operation": "ENCRYPTED_BACKUP",
            "original": "$(basename $backup_file)",
            "encrypted": "$(basename $encrypted_file)",
            "size_bytes": $encrypted_size,
            "encryption": "AES256_OpenSSL",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
                    echo "        • Encrypted: $(basename $backup_file) → $(basename $encrypted_file)"
                fi
            fi
        done

        # Secure the encryption key
        chmod 400 "$encryption_key"
        echo "      ✓ Encryption key: $encryption_key (keep secure!)"

    else
        echo "      ⚠ Encryption tools not available - skipping encrypted backup"
    fi
}

create_cloud_backup() {
    local report_file="$1"
    local backup_dir="$2"

    echo "     • Preparing cloud backup..."

    # Check for cloud storage locations
    local cloud_locations=(
        "$HOME/Library/Mobile Documents/com~apple~CloudDocs"
        "$HOME/Google Drive"
        "$HOME/Dropbox"
        "$HOME/OneDrive"
    )

    for cloud_dir in "${cloud_locations[@]}"; do
        if [[ -d "$cloud_dir" ]]; then
            local cloud_backup_dir="${cloud_dir}/Backups/QuantumNexus"
            mkdir -p "$cloud_backup_dir"

            # Copy critical backups to cloud
            for backup_file in "$backup_dir"/*.tar.gz; do
                if [[ -f "$backup_file" ]] && [[ $(stat -f "%z" "$backup_file" 2>/dev/null || stat -c "%s" "$backup_file")
-lt 100000000 ]]; then
                    cp "$backup_file" "${cloud_backup_dir}/"
                    echo "        • Copied to cloud: $(basename $backup_file) → $(basename $cloud_dir)"

                    cat >> "$report_file" << EOF
        {
            "operation": "CLOUD_BACKUP",
            "file": "$(basename $backup_file)",
            "cloud_provider": "$(basename $cloud_dir)",
            "cloud_path": "$cloud_backup_dir",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
```

```
            fi
        done
    fi
done

# Create cloud sync script
cat > "$backup_dir/cloud_sync.sh" << 'EOF'
#!/bin/bash
# Cloud Sync Script for Dotfile Backups
# Configure your cloud storage paths below

CLOUD_PATHS=(
    "$HOME/Library/Mobile Documents/com~apple~CloudDocs/Backups"
    "$HOME/Google Drive/Backups"
    "$HOME/Dropbox/Backups"
    "$HOME/OneDrive/Backups"
)

BACKUP_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
LOG_FILE="${BACKUP_DIR}/cloud_sync_$(date +%Y%m%d).log"

log_message() {
    echo "[$(date)] $1" >> "$LOG_FILE"
}

sync_to_cloud() {
    for cloud_path in "${CLOUD_PATHS[@]}"; do
        if [[ -d "$cloud_path" ]]; then
            log_message "Syncing to: $cloud_path"
            rsync -av --delete \
                --exclude="*.key" \
                --exclude="*.enc" \
                --exclude="*.gpg" \
                "$BACKUP_DIR/" "$cloud_path/QuantumNexus/" >> "$LOG_FILE" 2>&1
        fi
    done
}

# Daily sync
sync_to_cloud
log_message "Cloud sync completed"
EOF

    chmod +x "$backup_dir/cloud_sync.sh"
    echo "      ✓ Cloud sync script created: $backup_dir/cloud_sync.sh"
}

audit_dotfile_security() {
    echo "  • Performing dotfile security audit..."

    local security_report="${QNEXUS_REPORTS}/dotfile_security_$(date +%Y%m%d_%H%M%S).json"
    local vulnerabilities_found=0

    cat > "$security_report" << EOF
{
    "audit_timestamp": "$(date -u -Iseconds)",
    "vulnerabilities": [
EOF

    # Check 1: World-writable dotfiles
    echo "    • Checking for insecure permissions..."
    find "$HOME" -maxdepth 2 -name ".*" -type f -perm -o+w 2>/dev/null | while read -r file; do
        local perms=$(stat -f "%Sp" "$file" 2>/dev/null || stat -c "%A" "$file")

        cat >> "$security_report" << EOF
        {
            "type": "INSECURE_PERMISSIONS",
            "severity": "HIGH",
            "file": "$file",
            "permissions": "$perms",
            "issue": "World-writable dotfile",
            "recommendation": "Run: chmod o-w '$file'",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
        ((vulnerabilities_found++))
        echo "      ⚠ Insecure permissions: $file ($perms)"
    done

    # Check 2: Dotfiles containing sensitive information
    echo "    • Checking for exposed secrets..."
    check_for_exposed_secrets "$security_report"

    # Check 3: Suspicious dotfiles
    echo "    • Checking for suspicious dotfiles..."
    check_suspicious_dotfiles "$security_report"

    # Check 4: SSH key permissions
    echo "    • Checking SSH key security..."
    check_ssh_security "$security_report"

    # Check 5: AWS credential security
    echo "    • Checking AWS credential security..."
    check_aws_security "$security_report"

    # Remove trailing comma
    sed -i '' '$ s/,$//' "$security_report" 2>/dev/null || sed -i '$ s/,$//' "$security_report"

    cat >> "$security_report" << EOF
    ],
    "summary": {
        "total_vulnerabilities": $vulnerabilities_found,
        "high_severity": $(grep -c '"severity": "HIGH"' "$security_report"),
        "medium_severity": $(grep -c '"severity": "MEDIUM"' "$security_report"),
        "low_severity": $(grep -c '"severity": "LOW"' "$security_report"),
        "security_score": "$(calculate_security_score $vulnerabilities_found)"
    }
}
EOF

    echo ""
    echo "    ✓ Dotfile security audit complete:"
    echo "      - Vulnerabilities found: $vulnerabilities_found"
    echo "      - Security score: $(calculate_security_score $vulnerabilities_found)/100"
    echo "      - Report: $security_report"

    quantum_log "SECURITY" "Dotfile security audit completed: $vulnerabilities_found vulnerabilities found"
}

check_for_exposed_secrets() {
    local report_file="$1"

    # Patterns that indicate possible secrets
    local secret_patterns=(
        "AKIA[0-9A-Z]{16}"                # AWS Access Key ID
        "[0-9a-zA-Z/+]{40}"               # AWS Secret Key
        "sk_live_[0-9a-zA-Z]{24}"         # Stripe Secret Key
        "sk_test_[0-9a-zA-Z]{24}"         # Stripe Test Key
        "password[[:space:]]*=[[:space:]]*[\"]?[^\"]+[\"]?"
        "api_key[[:space:]]*=[[:space:]]*[\"]?[^\"]+[\"]?"
        "secret[[:space:]]*=[[:space:]]*[\"]?[^\"]+[\"]?"
        "token[[:space:]]*=[[:space:]]*[\"]?[^\"]+[\"]?"
    )

    find "$HOME" -maxdepth 3 -name ".*" -type f -exec grep -l -E "$(IFS="|"; echo "${secret_patterns[*]}")" {} \;
2>/dev/null | \
    while read -r file; do
        # Skip binary files
        if file "$file" | grep -q "text"; then
            local matches=$(grep -n -E "$(IFS="|"; echo "${secret_patterns[*]}")" "$file" | head -3)

            cat >> "$report_file" << EOF
        {
            "type": "EXPOSED_SECRET",
            "severity": "CRITICAL",
            "file": "$file",
            "matches": "$(echo $matches | sed 's/"/\\"/g')",
            "recommendation": "Remove or encrypt sensitive information",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      ⚠ Exposed secret in: $file"
        fi
    done
}

check_suspicious_dotfiles() {
    local report_file="$1"

    # Suspicious dotfile patterns
    local suspicious_patterns=(
        ".*\\.sh" ".*\\.py" ".*\\.js" ".*\\.php"  # Executable scripts as dotfiles
        "^\\._"                                   # macOS resource fork files
        "\\.DS_Store$"                            # macOS directory metadata
    )

    find "$HOME" -maxdepth 2 -name ".*" -type f | while read -r file; do
        local filename=$(basename "$file")

        for pattern in "${suspicious_patterns[@]}"; do
            if [[ "$filename" =~ $pattern ]]; then
```

```
                cat >> "$report_file" << EOF
        {
            "type": "SUSPICIOUS_FILE",
            "severity": "MEDIUM",
            "file": "$file",
            "pattern": "$pattern",
            "recommendation": "Review and remove if unnecessary",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
                echo "      ⚠ Suspicious dotfile: $filename"
                break
            fi
        done
    done
}

check_ssh_security() {
    local report_file="$1"

    if [[ -d "$HOME/.ssh" ]]; then
        # Check SSH directory permissions
        local ssh_dir_perms=$(stat -f "%Sp" "$HOME/.ssh" 2>/dev/null || stat -c "%A" "$HOME/.ssh")
        if [[ "$ssh_dir_perms" != "drwx------" ]]; then
            cat >> "$report_file" << EOF
        {
            "type": "INSECURE_SSH_DIR",
            "severity": "HIGH",
            "directory": "$HOME/.ssh",
            "permissions": "$ssh_dir_perms",
            "recommendation": "Run: chmod 700 '$HOME/.ssh'",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      ⚠ Insecure SSH directory permissions: $ssh_dir_perms"
        fi

        # Check private key permissions
        find "$HOME/.ssh" -name "id_*" ! -name "*.pub" -type f | while read -r key; do
            local key_perms=$(stat -f "%Sp" "$key" 2>/dev/null || stat -c "%A" "$key")
            if [[ "$key_perms" != "-rw-------" ]]; then
                cat >> "$report_file" << EOF
        {
            "type": "INSECURE_SSH_KEY",
            "severity": "CRITICAL",
            "key": "$key",
            "permissions": "$key_perms",
            "recommendation": "Run: chmod 600 '$key'",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
                echo "      ⚠ Insecure SSH key permissions: $key ($key_perms)"
            fi
        done
    fi
}
```

```
check_aws_security() {
    local report_file="$1"

    if [[ -f "$HOME/.aws/credentials" ]]; then
        local creds_perms=$(stat -f "%Sp" "$HOME/.aws/credentials" 2>/dev/null || stat -c "%A" "$HOME/.aws/
credentials")
        if [[ "$creds_perms" != "-rw-------" ]]; then
            cat >> "$report_file" << EOF
        {
            "type": "INSECURE_AWS_CREDENTIALS",
            "severity": "CRITICAL",
            "file": "$HOME/.aws/credentials",
            "permissions": "$creds_perms",
            "recommendation": "Run: chmod 600 '$HOME/.aws/credentials'",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      ⚠ Insecure AWS credentials permissions: $creds_perms"
        fi
    fi

    if [[ -f "$HOME/.aws/config" ]]; then
        local config_perms=$(stat -f "%Sp" "$HOME/.aws/config" 2>/dev/null || stat -c "%A" "$HOME/.aws/
config")
        if [[ "$config_perms" =~ "w.+" ]]; then
            cat >> "$report_file" << EOF
        {
            "type": "INSECURE_AWS_CONFIG",
            "severity": "HIGH",
            "file": "$HOME/.aws/config",
            "permissions": "$config_perms",
            "recommendation": "Run: chmod 600 '$HOME/.aws/config'",
            "timestamp": "$(date -u -Iseconds)"
        },
EOF
            echo "      ⚠ Insecure AWS config permissions: $config_perms"
        fi
    fi
}

calculate_security_score() {
    local vulnerabilities=$1
    local score=100

    # Deduct points based on vulnerabilities
    if [[ $vulnerabilities -gt 10 ]]; then
        score=30
    elif [[ $vulnerabilities -gt 5 ]]; then
        score=50
    elif [[ $vulnerabilities -gt 2 ]]; then
        score=70
    elif [[ $vulnerabilities -gt 0 ]]; then
        score=85
    fi

    # Check for critical vulnerabilities (additional deduction)
    if [[ -f "$security_report" ]] && grep -q '"severity": "CRITICAL"' "$security_report"; then
```

```
        score=$((score - 20))
    fi

    # Ensure score is between 0 and 100
    if [[ $score -lt 0 ]]; then
        score=0
    elif [[ $score -gt 100 ]]; then
        score=100
    fi

    echo $score
}

# Add to main controller
if [[ "${BASH_SOURCE[0]}" == "${0}" ]] || [[ "${ZSH_EVAL_CONTEXT}" == "toplevel" ]]; then
    # Initialize environment
    source "${QNEXUS_MODULES}/init.zsh"

    # Run dotfile management
    manage_dotfiles
fi
```