

Напиши питоновский скрипт который соберет данные о системе, дисках, разделах и тд и поможет обновить винду с 10 до 11 или уточнит информацию узнает mbr gpt еще и тд свободное место и тд

Думал на протяжении 31s >

Ниже — готовый скрипт **Windows 10/11**, который собирает инфу о системе/дисках/разделах, определяет **MBR/GPT, UEFI/BIOS, Secure Boot, TPM (и версию 2.0)**, считает **RAM/CPU/место на дисках**, и в конце даёт **чек-лист готовности к апгрейду до Windows 11**.

(Работает без сторонних библиотек, через PowerShell. Лучше запускать **от имени администратора**, тогда Secure Boot/TPM читаются стабильнее.)

python

```
# win11_readiness_report.py
# Windows 10/11 system + disk + partition inventory, plus Windows 11 readiness ch
# Usage:
#   python win11_readiness_report.py
#   python win11_readiness_report.py --outdir .
# Tip: run in elevated terminal (Run as Administrator) for best results.
```

```
import argparse
import json
import os
import platform
import re
import socket
import subprocess
import sys
from datetime import datetime
from typing import Any, Dict, List, Optional, Tuple, Union
```

```
JsonLike = Union[Dict[str, Any], List[Any], str, int, float, bool, None]
```

```

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
    """Run command and return (code, stdout, stderr)."""
    p = subprocess.run(cmd, capture_output=True, text=True, encoding="utf-8", err
    return p.returncode, p.stdout.strip(), p.stderr.strip()

def run_powershell(ps_script: str) -> Tuple[int, str, str]:
    """Run PowerShell script."""
    cmd = [
        "powershell",
        "-NoProfile",
        "-ExecutionPolicy",
        "Bypass",
        "-Command",
        ps_script,
    ]
    return run_cmd(cmd)

def ps_json(ps_script: str) -> Tuple[Optional[JsonLike], Optional[str]]:
    """
    Run PowerShell pipeline, forcing JSON output.
    Returns (parsed_json, error_str).
    """
    code, out, err = run_powershell(ps_script + " | ConvertTo-Json -Depth 6")
    if code != 0:
        return None, err or out or f"PowerShell exited with code {code}"
    if not out:
        return None, "Empty PowerShell output"
    # Some systems output weird leading chars; strip BOM-like junk.
    out_clean = out.lstrip("\ufeff").strip()
    try:
        return json.loads(out_clean), None
    except Exception as e:
        return None, f"JSON parse error: {e}\nRaw output:\n{out[:1000]}"

def ensure_list(x: JsonLike) -> List[Any]:
    if x is None:
        return []
    if isinstance(x, list):

```

```
        return x
    return [x]

def bytes_to_gb(b: Optional[Union[int, float]]) -> Optional[float]:
    if b is None:
        return None
    try:
        return round(float(b) / (1024 ** 3), 2)
    except Exception:
        return None

def pretty_bool(x: Any) -> str:
    if x is True:
        return "YES"
    if x is False:
        return "NO"
    return "UNKNOWN"

def safe_get(d: Any, key: str, default=None):
    if isinstance(d, dict):
        return d.get(key, default)
    return default

def collect_basic_system() -> Dict[str, Any]:
    info: Dict[str, Any] = {}
    info["hostname"] = socket.gethostname()
    info["user"] = os.environ.get("USERNAME") or os.environ.get("USER")
    info["system_drive"] = os.environ.get("SystemDrive", "C:")
    info["platform"] = {
        "system": platform.system(),
        "release": platform.release(),
        "version": platform.version(),
        "machine": platform.machine(),
        "processor": platform.processor(),
    }
    # Windows build info (best-effort)
    # Get-ComputerInfo has a lot; pull only a few useful fields.
    ps = r"""
    $ci = Get-ComputerInfo
```

```

[pscustomobject]@{
    WindowsProductName = $ci.WindowsProductName
    WindowsVersion      = $ci.WindowsVersion
    OsBuildNumber       = $ci.OsBuildNumber
    OsHardwareAbstractionLayer = $ci.OsHardwareAbstractionLayer
    WindowsEditionId    = $ci.WindowsEditionId
    CsManufacturer      = $ci.CsManufacturer
    CsModel             = $ci.CsModel
}
"""

data, err = ps_json(ps)
info["windows"] = data if data is not None else {"error": err}
return info


def collect_cpu_ram() -> Dict[str, Any]:
    ps = r"""
    $cpu = Get-CimInstance Win32_Processor | Select-Object Name, Manufacturer, Nu
    $cs  = Get-CimInstance Win32_ComputerSystem | Select-Object TotalPhysicalMemo
    [pscustomobject]@{
        CPU = $cpu
        TotalPhysicalMemory = $cs.TotalPhysicalMemory
    }
    """

    data, err = ps_json(ps)
    if data is None:
        return {"error": err}
    # Normalize
    cpu_list = ensure_list(data.get("CPU"))
    total_mem = data.get("TotalPhysicalMemory")
    return {
        "cpu": cpu_list,
        "ram_bytes": total_mem,
        "ram_gb": bytes_to_gb(total_mem),
    }


def collect_firmware_secureboot_tpm() -> Dict[str, Any]:
    info: Dict[str, Any] = {}

    # Firmware type (1 BIOS, 2 UEFI)
    ps_fw = r"(Get-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control' -N
    code, out, err = run_powershell(ps_fw)

```

```
fw = None
if code == 0 and out:
    try:
        fw = int(out.strip())
    except Exception:
        fw = None
info["firmware"] = {
    "pe_firmware_type_raw": out if out else None,
    "pe_firmware_type": fw,
    "boot_mode": ("UEFI" if fw == 2 else ("BIOS/Legacy" if fw == 1 else "UNKNOWN"),
    "error": None if code == 0 else (err or out),
}

# Secure Boot (needs UEFI and often admin)
ps_sb = r"Confirm-SecureBootUEFI"
code, out, err = run_powershell(ps_sb)
sb_val: Optional[bool] = None
sb_err: Optional[str] = None
if code == 0 and out:
    if out.strip().lower() == "true":
        sb_val = True
    elif out.strip().lower() == "false":
        sb_val = False
    else:
        sb_val = None
else:
    sb_err = err or out or f"PowerShell exited with code {code}"
info["secure_boot"] = {
    "enabled": sb_val,
    "raw": out if out else None,
    "error": sb_err,
}

# TPM
# Get-Tpm + Win32_Tpm for SpecVersion
ps_tpm = r""
$tpm1 = $null
try { $tpm1 = Get-Tpm } catch { $tpm1 = $null }
$tpm2 = $null
try {
    $tpm2 = Get-CimInstance -Namespace root\cimv2\security\microsofttpm -ClassName Win32_Tpm
    Select-Object SpecVersion, ManufacturerIdTxt, ManufacturerVersion,
} catch { $tpm2 = $null }
```

```
[pscustomobject]@{
    GetTpm = $tpm1
    Win32Tpm = $tpm2
}
"""

data, err = ps_json(ps_tpm)
if data is None:
    info["tpm"] = {"error": err}
    return info

gettpm = data.get("GetTpm")
win32tpm = data.get("Win32Tpm")
win32_list = ensure_list(win32tpm)

# Determine TPM 2.0 from SpecVersion like "2.0,1.2"
spec_versions = []
for item in win32_list:
    sv = safe_get(item, "SpecVersion")
    if sv:
        spec_versions.append(str(sv))

tpm2_supported = None
if spec_versions:
    # If any spec contains "2.0"
    tpm2_supported = any(re.search(r"\b2\.0\b", sv) for sv in spec_versions)

info["tpm"] = {
    "gettpm": gettpm,
    "win32tpm": win32_list,
    "spec_versions": spec_versions,
    "tpm2_supported": tpm2_supported,
    "error": None,
}
return info


def collect_disks_partitions_volumes() -> Dict[str, Any]:
    info: Dict[str, Any] = {}

    # Disks
    ps_disks = r"Get-Disk | Select-Object Number, FriendlyName, SerialNumber, Bus
disks, err = ps_json(ps_disks)
if disks is None:
```

```

        info["disks_error"] = err
        disks_list: List[Any] = []
    else:
        disks_list = ensure_list(disks)

# Partitions
ps_parts = r"Get-Partition | Select-Object DiskNumber, PartitionNumber, Drive
parts, err2 = ps_json(ps_parts)
if parts is None:
    info["partitions_error"] = err2
    parts_list: List[Any] = []
else:
    parts_list = ensure_list(parts)

# Volumes
ps_vols = r"Get-Volume | Select-Object DriveLetter, FileSystemLabel, FileSyst
vols, err3 = ps_json(ps_vols)
if vols is None:
    info["volumes_error"] = err3
    vols_list: List[Any] = []
else:
    vols_list = ensure_list(vols)

# Attach GB conversions
for d in disks_list:
    d["SizeGB"] = bytes_to_gb(d.get("Size"))
for p in parts_list:
    p["SizeGB"] = bytes_to_gb(p.get("Size"))
for v in vols_list:
    v["SizeGB"] = bytes_to_gb(v.get("Size"))
    v["FreeGB"] = bytes_to_gb(v.get("SizeRemaining"))

info["disks"] = disks_list
info["partitions"] = parts_list
info["volumes"] = vols_list

return info

def find_system_disk_partitionstyle(report: Dict[str, Any]) -> Tuple[Optional[int
"""
    Try to determine which disk contains the SystemDrive (e.g. C:)
    by matching volumes/partitions and get its PartitionStyle.

```

```
"""
system_drive = report.get("basic", {}).get("system_drive", "C:")
drive_letter = system_drive.replace(":", "").upper()

parts = report.get("storage", {}).get("partitions", [])
disks = report.get("storage", {}).get("disks", [])

disk_number = None
for p in parts:
    dl = p.get("DriveLetter")
    if dl and str(dl).upper() == drive_letter:
        disk_number = p.get("DiskNumber")
        break

if disk_number is None:
    # fallback: find disk marked IsSystem True
    for d in disks:
        if d.get("IsSystem") is True:
            disk_number = d.get("Number")
            break

partition_style = None
if disk_number is not None:
    for d in disks:
        if d.get("Number") == disk_number:
            partition_style = d.get("PartitionStyle")
            break

return disk_number, partition_style

def assess_windows11_readiness(report: Dict[str, Any]) -> Dict[str, Any]:
    checks: List[Dict[str, Any]] = []
    verdict = "UNKNOWN"

    # CPU 64-bit
    cpu_info = report.get("hw", {}).get("cpu", [])
    addr_widths = []
    for c in cpu_info:
        aw = c.get("AddressWidth")
        if aw is not None:
            addr_widths.append(aw)
    cpu_64 = True if (addr_widths and all(int(x) >= 64 for x in addr_widths)) else
```



```
checks.append({
    "check": "CPU 64-bit",
    "required": "Yes",
    "value": "64-bit" if cpu_64 else ("Not 64-bit" if cpu_64 is False else "U
    "pass": True if cpu_64 else False if cpu_64 is False else None,
})

# RAM >= 4 GB
ram_gb = report.get("hw", {}).get("ram_gb")
ram_ok = None
if ram_gb is not None:
    ram_ok = ram_gb >= 4.0
checks.append({
    "check": "RAM",
    "required": ">= 4 GB",
    "value": f"{ram_gb} GB" if ram_gb is not None else "Unknown",
    "pass": ram_ok,
})

# Storage: system volume total >= 64GB and free >= 25GB (practical)
system_drive = report.get("basic", {}).get("system_drive", "C:")
sys_letter = system_drive.replace(":", "").upper()

volumes = report.get("storage", {}).get("volumes", [])
sys_vol = None
for v in volumes:
    dl = v.get("DriveLetter")
    if dl and str(dl).upper() == sys_letter:
        sys_vol = v
        break

total_gb = sys_vol.get("SizeGB") if sys_vol else None
free_gb = sys_vol.get("FreeGB") if sys_vol else None

storage_ok = None
if total_gb is not None:
    storage_ok = total_gb >= 64.0
free_ok = None
if free_gb is not None:
    free_ok = free_gb >= 25.0

checks.append({
```

```
"check": "System drive capacity",
"required": ">= 64 GB (Windows 11 requirement)",
"value": f"{total_gb} GB" if total_gb is not None else "Unknown",
"pass": storage_ok,
})
checks.append({
    "check": "System drive free space",
    "required": ">= 25 GB (recommended for upgrade headroom)",
    "value": f"{free_gb} GB free" if free_gb is not None else "Unknown",
    "pass": free_ok,
})

# Firmware UEFI
boot_mode = report.get("security", {}).get("firmware", {}).get("boot_mode")
uefi_ok = True if boot_mode == "UEFI" else (False if boot_mode in ("BIOS/Lega
checks.append({
    "check": "Boot mode",
    "required": "UEFI",
    "value": boot_mode or "Unknown",
    "pass": uefi_ok,
})

# Secure Boot
sb = report.get("security", {}).get("secure_boot", {}).get("enabled")
sb_ok = sb if isinstance(sb, bool) else None
checks.append({
    "check": "Secure Boot",
    "required": "Enabled",
    "value": pretty_bool(sb),
    "pass": sb_ok,
    "note": report.get("security", {}).get("secure_boot", {}).get("error"),
})

# TPM 2.0
tpm = report.get("security", {}).get("tpm", {})
tpm2_supported = tpm.get("tpm2_supported")
gettpm = tpm.get("gettpm") if isinstance(tpm, dict) else None
tpm_present = safe_get(gettpm, "TpmPresent") if gettpm else None
tpm_ready = safe_get(gettpm, "TpmReady") if gettpm else None

# Consider OK if present+ready AND spec says 2.0 (if known)
tpm_ok = None
if isinstance(tpm_present, bool) and isinstance(tpm_ready, bool):
```

```

    if tpm2_supported is True:
        tpm_ok = (tpm_present and tpm_ready)
    elif tpm2_supported is False:
        tpm_ok = False
    else:
        # no spec version; still report presence
        tpm_ok = (tpm_present and tpm_ready)

checks.append({
    "check": "TPM",
    "required": "TPM 2.0 present & ready",
    "value": f"Present={pretty_bool(tpm_present)}, Ready={pretty_bool(tpm_rea
    "pass": tpm_ok,
    "note": tpm.get("error") if isinstance(tpm, dict) else None,
})

# GPT vs MBR for system disk (practical: UEFI usually wants GPT)
disk_no, part_style = find_system_disk_partitionstyle(report)
gpt_ok = None
if part_style:
    gpt_ok = True if str(part_style).upper() == "GPT" else False if str(part_

checks.append({
    "check": "System disk partition style",
    "required": "GPT (recommended/typical for UEFI + Win11)",
    "value": f"Disk {disk_no}: {part_style}" if disk_no is not None else (par
    "pass": gpt_ok,
})

# Decide verdict:
# If any explicit False among key checks -> NOT READY.
key_passes = []
for c in checks:
    if c["pass"] is not None:
        key_passes.append(c["pass"])
if any(p is False for p in key_passes):
    verdict = "NOT READY (some requirements failed)"
elif key_passes and all(p is True for p in key_passes):
    verdict = "READY (requirements look OK)"
else:
    verdict = "PARTIAL/UNKNOWN (some checks couldn't be confirmed)"

# Provide next steps guidance based on common blockers

```

```

actions: List[str] = []
if uefi_ok is False:
    actions.append("Boot mode is Legacy/BIOS. Windows 11 typically requires U
if gpt_ok is False:
    actions.append("System disk is MBR. For UEFI/Windows 11 you may need GPT.
if sb_ok is False:
    actions.append("Secure Boot is OFF. Enable it in UEFI settings (may requi
if tpm_ok is False:
    actions.append("TPM requirement failed. Enable TPM in BIOS/UEFI (Intel PT
if free_ok is False:
    actions.append("Not enough free space on system drive. Aim for 25-30 GB f
if storage_ok is False:
    actions.append("System drive total capacity below 64 GB. Windows 11 requi

return {"verdict": verdict, "checks": checks, "suggested_actions": actions}

def render_text_report(report: Dict[str, Any]) -> str:
    lines: List[str] = []
    basic = report.get("basic", {})
    win = basic.get("windows", {})
    hw = report.get("hw", {})
    sec = report.get("security", {})
    storage = report.get("storage", {})
    assess = report.get("assessment", {})

    lines.append("=== Windows 11 Readiness / System Inventory Report ===")
    lines.append(f"Generated: {report.get('generated_at')}")
    lines.append(f"Host: {basic.get('hostname')} | User: {basic.get('user')} | Sy
    lines.append("")

    lines.append("--- Windows ---")
    if isinstance(win, dict) and "error" not in win:
        lines.append(f"Product: {win.get('WindowsProductName')}")
        lines.append(f"Version: {win.get('WindowsVersion')} | Build: {win.get('Os
        lines.append(f"Manufacturer/Model: {win.get('CsManufacturer')} {win.get('
    else:
        lines.append(f"Windows info error: {win}")
    lines.append("")

    lines.append("--- CPU / RAM ---")
    cpu_list = hw.get("cpu", [])
    if cpu_list:

```

```

        for i, c in enumerate(cpu_list, start=1):
            lines.append(f"CPU #{i}: {c.get('Name')} | Cores: {c.get('NumberOfCores')}")
        else:
            lines.append("CPU: Unknown")
    lines.append(f"RAM: {hw.get('ram_gb')} GB")
    lines.append("")

    lines.append("--- Firmware / Security ---")
    fw = sec.get("firmware", {})
    lines.append(f"Boot mode: {fw.get('boot_mode')} (PEFirmwareType={fw.get('pe_firmware_type')})")
    sb = sec.get("secure_boot", {})
    lines.append(f"Secure Boot: {pretty_bool(sb.get('enabled'))}" + (f" | note: {sb.get('note')}" if sb.get('note') else ""))
    tpm = sec.get("tpm", {})
    if isinstance(tpm, dict) and "error" not in tpm:
        gettpm = tpm.get("gettpm") or {}
        lines.append(
            "TPM: "
            f"Present={pretty_bool(safe_get(gettpm, 'TpmPresent'))}, "
            f"Ready={pretty_bool(safe_get(gettpm, 'TpmReady'))}, "
            f"Enabled={pretty_bool(safe_get(gettpm, 'TpmEnabled'))}, "
            f"Activated={pretty_bool(safe_get(gettpm, 'TpmActivated'))}, "
            f"TPM2={pretty_bool(tpm.get('tpm2_supported'))}, "
            f"SpecVersions={tpm.get('spec_versions')}"
        )
    else:
        lines.append(f"TPM info: {tpm}")
    lines.append("")

    lines.append("--- Disks ---")
    disks = storage.get("disks", [])
    if disks:
        for d in disks:
            lines.append(
                f"Disk {d.get('Number')}: {d.get('FriendlyName')} | {d.get('SizeGB')} GB | "
                f"Style={d.get('PartitionStyle')} | Bus={d.get('BusType')} | Media={d.get('Media')} | "
                f"IsSystem={d.get('IsSystem')} IsBoot={d.get('IsBoot')}"
            )
    else:
        lines.append("No disk data (PowerShell error?)")
        if storage.get("disks_error"):
            lines.append(f"Disks error: {storage.get('disks_error')}")
    lines.append("")

```

```

lines.append("--- Volumes ---")
vols = storage.get("volumes", [])
if vols:
    for v in vols:
        dl = v.get("DriveLetter")
        if not dl:
            continue
        lines.append(
            f"{dl}: | FS={v.get('FileSystemType')} | Label={v.get('FileSystem')} | Size={v.get('SizeGB')} GB | Free={v.get('FreeGB')} GB | Health="
        )
else:
    lines.append("No volume data")
    if storage.get("volumes_error"):
        lines.append(f"Volumes error: {storage.get('volumes_error')}")
lines.append("")

lines.append("=== Windows 11 Readiness Checks ===")
lines.append(f"Verdict: {assess.get('verdict')}")
lines.append("")
for c in assess.get("checks", []):
    status = "PASS" if c["pass"] is True else ("FAIL" if c["pass"] is False else "UNKNOWN")
    note = f" | note: {c.get('note')}" if c.get("note") else ""
    lines.append(f"- [{status}] {c['check']}: {c['value']} (required: {c['required']}){note}")

actions = assess.get("suggested_actions", [])
if actions:
    lines.append("")
    lines.append("Next steps (based on blockers):")
    for a in actions:
        lines.append(f" * {a}")

lines.append("")
lines.append("Practical tip: Microsoft 'PC Health Check' is the official final check for Windows 11 readiness.")
lines.append("This script focuses on the usual hard blockers: TPM/SecureBoot/BitLocker.")
return "\n".join(lines)

def main():
    if platform.system().lower() != "windows":
        print("This script is intended for Windows 10/11 only.")
        sys.exit(1)

```

```
ap = argparse.ArgumentParser(description="Collect system/disk/security info a
ap.add_argument("--outdir", default=".", help="Directory to save report files
args = ap.parse_args()

outdir = os.path.abspath(args.outdir)
os.makedirs(outdir, exist_ok=True)

report: Dict[str, Any] = {
    "generated_at": datetime.now().isoformat(timespec="seconds"),
    "basic": collect_basic_system(),
    "hw": collect_cpu_ram(),
    "security": collect_firmware_secureboot_tpm(),
    "storage": collect_disks_partitions_volumes(),
}
report["assessment"] = assess_windows11_readiness(report)

text = render_text_report(report)

host = report["basic"].get("hostname", "host")
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base_name = f"win11_readiness_{host}_{ts}"

json_path = os.path.join(outdir, base_name + ".json")
txt_path = os.path.join(outdir, base_name + ".txt")

with open(json_path, "w", encoding="utf-8") as f:
    json.dump(report, f, ensure_ascii=False, indent=2)

with open(txt_path, "w", encoding="utf-8") as f:
    f.write(text)

print(text)
print(f"\nSaved:\n- {txt_path}\n- {json_path}")

if __name__ == "__main__":
    main()
```

 Копировать код

Как запускать (быстро и без магии)

1. Сохрани как win11_readiness_report.py

FullyQualifiedErrorId : EmptyPipeElement'}

--- CPU / RAM ---

CPU: Unknown

RAM: None GB

--- Firmware / Security ---

Boot mode: UNKNOWN (PEFirmwareType=None)

Secure Boot: UNKNOWN | note: Confirm-SecureBootUEFI :

?? 烈????? ?????????? ?????室?騾

?? ???????. ? ??????棘 ?≅?????

??□?:1 ??????:1

+ Confirm-SecureBootUEFI

+ ~~~~~~

+ CategoryInfo : PermissionDenied:

(Microsoft.Secur...BootUefiCommand:ConfirmSecureBootUefiCommand) [Conf

irm-SecureBootUEFI], UnauthorizedAccessException

+ FullyQualifiedErrorId :

SetPrivilegeFailed,Microsoft.SecureBoot.Commands.ConfirmSecureBootUefiCommand

TPM info: {'error': '??□?:13 ??????:6\n+ |

ConvertTo-Json -Depth 6\n+ ~\n?????↪

????????? ?????????? ???? ??????□?????.\n +

CategoryInfo : ParserError: (:) [],

ParentContainsErrorRecordException\n +

FullyQualifiedErrorId : EmptyPipeElement'}

--- Disks ---

Disk 0: P4-240 | 223.57 GB | Style=GPT | Bus=SATA |

Media=None | IsSystem=True IsBoot=True

Disk 1: SAMSUNG HD161GJ | 149.05 GB | Style=MBR |

Bus=SATA | Media=None | IsSystem=False IsBoot=False

--- Volumes ---

G: | FS=NTFS | Label= | Size=24.04 GB | Free=17.57 GB | Health=Healthy

E: | FS=NTFS | Label=Win 7 | Size=30.01 GB | Free=29.93 GB | Health=Healthy

F: | FS=NTFS | Label= | Size=95.0 GB | Free=49.87 GB | Health=Healthy

C: | FS=NTFS | Label= | Size=111.4 GB | Free=6.74 GB | Health=Healthy

* Not enough free space on system drive. Aim for 25–30 GB free before upgrading (cleanup, move data, uninstall junk).

Practical tip: Microsoft 'PC Health Check' is the official final judge for CPU support list quirks.

This script focuses on the usual hard blockers:

TPM/SecureBoot/UEFI/GPT/space/RAM/64-bit.

Saved:

- C:\Users\Славиус\Desktop\win11_readiness_DESKTOP-1OAT63D_20251227_140507.txt

- C:\Users\Славиус\Desktop\win11_readiness_DESKTOP-1OAT63D_20251227_140507.json

powershell отчет:

Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Users\Славиус> python .\win11_readiness_report.py  
--outdir .
```

```
>> python .\win11_readiness_report.py --outdir .
```

```
C:\Users\Славиус\AppData\Local\Programs\Python\Python  
n313\python.exe: can't open file
```

```
'C:\\Users\\Славиус\\win11_readiness_report.py': [Errno 2]  
No such file or directory
```

```
C:\Users\Славиус\AppData\Local\Programs\Python\Python  
n313\python.exe: can't open file
```

```
'C:\\Users\\Славиус\\win11_readiness_report.py': [Errno 2]  
No such file or directory
```

```
PS C:\Users\Славиус> python .\win11_readiness_report.py  
--outdir .
```

```
C:\Users\Славиус\AppData\Local\Programs\Python\Python  
n313\python.exe: can't open file
```

```
'C:\\Users\\Славиус\\win11_readiness_report.py': [Errno 2]  
No such file or directory
```

```
PS C:\Users\Славиус> python .\win11_readiness_report.py  
--outdir .
```

```
C:\Users\Славиус\AppData\Local\Programs\Python\Python
```

```
n313\python.exe: can't open file
'C:\\Users\\Славиус\\win11_readiness_report.py': [Errno 2]
No such file or directory
PS C:\\Users\\Славиус> python .\\win11_readiness_report.py
--outdir .
C:\\Users\\Славиус\\AppData\\Local\\Programs\\Python\\Pytho
n313\\python.exe: can't open file
'C:\\Users\\Славиус\\win11_readiness_report.py': [Errno 2]
No such file or directory
PS C:\\Users\\Славиус> python .\\win11_readiness_report.py
--outdir .
>>
C:\\Users\\Славиус\\AppData\\Local\\Programs\\Python\\Pytho
n313\\python.exe: can't open file
'C:\\Users\\Славиус\\win11_readiness_report.py': [Errno 2]
No such file or directory
PS C:\\Users\\Славиус> cd desktop
PS C:\\Users\\Славиус\\desktop> python
.\\win11_readiness_report.py --outdir .
>>
=== Windows 11 Readiness / System Inventory Report
===
Generated: 2025-12-27T14:07:02
Host: DESKTOP-1OAT63D | User: Славиус | SystemDrive: C:

--- Windows ---
Windows info error: {'error': '????:12 ????:6\\n+ |
ConvertTo-Json -Depth 6\\n+ ~\\n?????
?????????????????????????????????????\\n +
CategoryInfo : ParserError: (: [],
ParentContainsErrorRecordException\\n +
FullyQualifiedErrorId : EmptyPipeElement'}

--- CPU / RAM ---
CPU: Unknown
RAM: None GB

--- Firmware / Security ---
Boot mode: UNKNOWN (PEFirmwareType=None)
Secure Boot: UNKNOWN | note: Confirm-SecureBootUEFI :
????? ??????????????????????室?騾
?????????????. ? ??????棘 ?≡?????.
?????:1 ??????:1
```

```
+ Confirm-SecureBootUEFI
+ ~~~~~~
+ CategoryInfo          : PermissionDenied:
(Microsoft.Secur...BootUefiCommand:ConfirmSecureBootU
efiCommand) [Conf
irm-SecureBootUEFI], UnauthorizedAccessException
+ FullyQualifiedErrorId :
SetPrivilegeFailed,Microsoft.SecureBoot.Commands.Confir
mSecureBootUefiCommand
TPM info: {'error': '0x13 0x6\n+   |
ConvertTo-Json -Depth 6\n+   ~\n0x00000000
0x00000000 0x00000000 0x00000000 0x00000000.\n   +
CategoryInfo          : ParserError: (:) [],
ParentContainsErrorRecordException\n   +
FullyQualifiedErrorId : EmptyPipeElement'}
```

--- Disks ---

```
Disk 0: P4-240 | 223.57 GB | Style=GPT | Bus=SATA |
Media=None | IsSystem=True IsBoot=True
Disk 1: SAMSUNG HD161GJ | 149.05 GB | Style=MBR |
Bus=SATA | Media=None | IsSystem=False IsBoot=False
```

--- Volumes ---

```
G: | FS=NTFS | Label= | Size=24.04 GB | Free=17.57 GB |
Health=Healthy
E: | FS=NTFS | Label=Win 7 | Size=30.01 GB | Free=29.93
GB | Health=Healthy
F: | FS=NTFS | Label= | Size=95.0 GB | Free=49.87 GB |
Health=Healthy
C: | FS=NTFS | Label= | Size=111.4 GB | Free=6.73 GB |
Health=Healthy
D: | FS=NTFS | Label=0x00000000 | Size=111.4 GB |
Free=106.38 GB | Health=Healthy
H: | FS=Unknown | Label= | Size=0.0 GB | Free=0.0 GB |
Health=Healthy
```

=== Windows 11 Readiness Checks ===

Verdict: NOT READY (some requirements failed)

- [UNKNOWN] CPU 64-bit: Unknown (required: Yes)
- [UNKNOWN] RAM: Unknown (required: >= 4 GB)
- [PASS] System drive capacity: 111.4 GB (required: >= 64 GB (Windows 11 requirement))

```

- [FAIL] System drive free space: 6.73 GB free (required: >=
25 GB (recommended for upgrade headroom))
- [UNKNOWN] Boot mode: UNKNOWN (required: UEFI)
- [UNKNOWN] Secure Boot: UNKNOWN (required:
Enabled) | note: Confirm-SecureBootUEFI : ??? 烈????
????????????????室騾 ㄱ?????????. ??????棘
?≅?????.
??□?:1 ??????:1
+ Confirm-SecureBootUEFI
+ ~~~~~~
+ CategoryInfo      : PermissionDenied:
(Microsoft.Secur...BootUefiCommand:ConfirmSecureBootU
efiCommand) [Conf
irm-SecureBootUEFI], UnauthorizedAccessException
+ FullyQualifiedErrorId :
SetPrivilegeFailed,Microsoft.SecureBoot.Commands.Confir
mSecureBootUefiCommand
- [UNKNOWN] TPM: Present=UNKNOWN,
Ready=UNKNOWN, TPM2=UNKNOWN (required: TPM 2.0
present & ready) | note: ??□?:13 ??????:6
+ | ConvertTo-Json -Depth 6
+ ~
???→ ??????????????????
????□?????.
+ CategoryInfo      : ParserError: (:) [],
ParentContainsErrorRecordException
+ FullyQualifiedErrorId : EmptyPipeElement
- [PASS] System disk partition style: Disk 0: GPT (required:
GPT (recommended/typical for UEFI + Win11))

```

Next steps (based on blockers):

* Not enough free space on system drive. Aim for 25–30 GB free before upgrading (cleanup, move data, uninstall junk).

Practical tip: Microsoft 'PC Health Check' is the official final judge for CPU support list quirks.

This script focuses on the usual hard blockers:

TPM/SecureBoot/UEFI/GPT/space/RAM/64-bit.

Saved:

- C:\Users\Славиус\desktop\win11_readiness_DESKTOP-1OAT63D_20251227_140709.txt

```
- C:\Users\Славиус\desktop\win11_readiness_DESKTOP-1OAT63D_20251227_140709.json  
PS C:\Users\Славиус\desktop>
```

Думал на протяжении 1m 39s >

Виджу отчёт. Хорошие новости: **диск с системой уже GPT**, то есть половина боли (MBR→GPT) у тебя уже сделана.



Плохие новости: **C:** у тебя сейчас как студент перед зарплатой — **6.7 GB свободно**. Для апгрейда до Win11 это прям “не, дружище”.

И ещё: у скрипта выявился баг (мой 😊) — из-за него PowerShell-команды, которые должны были дать **CPU/RAM/TPM/Windows info**, падают с `EmptyPipeElement`, и поэтому эти поля стали `UNKNOWN`. Плюс русские ошибки у тебя “кракозябрами” из-за кодировки.

Ниже — исправленная версия (v2):

- фиксит `EmptyPipeElement` (оборачивает PowerShell в блок)
- нормально декодирует вывод (UTF-8 / cp866 / cp1251)
- чуть более надёжно вытягивает Boot mode / Secure Boot / TPM
- даёт тот же отчёт TXT+JSON

1) Что у тебя уже ясно по текущему отчёту

-  **Disk 0 (система): GPT, IsSystem=True, IsBoot=True — отлично.**
-  **C: 111.4 GB, Free 6.7 GB — главный стоппер.**
- **Disk 1: MBR** — это не страшно, пока система грузится с Disk 0.

Сначала освобождаем место, параллельно — запускаем новый скрипт от админа, чтобы понять TPM/SecureBoot/UEFI.

2) Скрипт v2 (просто замени файл и запусти)

Сохрани как `win11_readiness_report_v2.py` на Desktop и запусти.

```
python
```

 Копировать код

```
# win11_readiness_report_v2.py  
# Fixes:
```

```
# - no more "EmptyPipeElement" (wrap PS in scriptblock)
# - better decoding for RU Windows consoles (utf-8 / cp866 / cp1251)
# - more robust firmware detection
#
# Run (recommended as Admin):
# python .\win11_readiness_report_v2.py --outdir .

import argparse
import json
import os
import platform
import re
import socket
import subprocess
import sys
from datetime import datetime
from typing import Any, Dict, List, Optional, Tuple, Union

JsonLike = Union[Dict[str, Any], List[Any], str, int, float, bool, None]

def _decode_best_effort(b: bytes) -> str:
    if b is None:
        return ""
    for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
        try:
            return b.decode(enc)
        except Exception:
            continue
    return b.decode("utf-8", errors="replace")

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
    """Run command and return (code, stdout, stderr) with robust decoding."""
    p = subprocess.run(cmd, capture_output=True)
    out = _decode_best_effort(p.stdout).strip()
    err = _decode_best_effort(p.stderr).strip()
    return p.returncode, out, err

def run_powershell(ps_script: str) -> Tuple[int, str, str]:
    """
    Run PowerShell script.
    """
```


We force UTF8 output encoding inside PS to reduce mojibake.

"""

```
prefix = "[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new(); $Pro
cmd = ["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-Command",
return run_cmd(cmd)
```

```
def ps_json(ps_script: str) -> Tuple[Optional[JsonLike], Optional[str]]:
```

"""

Run PowerShell, capture objects, convert to JSON safely.

Key fix: wrap script in (& { ... }) so piping works even if script has newlin

"""

```
wrapped = f"(& {{ {ps_script} }}) | ConvertTo-Json -Depth 8 -Compress"
```

```
code, out, err = run_powershell(wrapped)
```

```
if code != 0:
```

```
    return None, err or out or f"PowerShell exited with code {code}"
```

```
if not out:
```

```
    return None, "Empty PowerShell output"
```

```
try:
```

```
    return json.loads(out), None
```

```
except Exception as e:
```

```
    return None, f"JSON parse error: {e}\nRaw output:\n{out[:1200]}"
```

```
def ensure_list(x: JsonLike) -> List[Any]:
```

```
    if x is None:
```

```
        return []
```

```
    if isinstance(x, list):
```

```
        return x
```

```
    return [x]
```

```
def bytes_to_gb(b: Optional[Union[int, float]]) -> Optional[float]:
```

```
    if b is None:
```

```
        return None
```

```
    try:
```

```
        return round(float(b) / (1024 ** 3), 2)
```

```
    except Exception:
```

```
        return None
```

```
def pretty_bool(x: Any) -> str:
```

```
    if x is True:
```

```
        return "YES"
    if x is False:
        return "NO"
    return "UNKNOWN"

def safe_get(d: Any, key: str, default=None):
    if isinstance(d, dict):
        return d.get(key, default)
    return default

def collect_basic_system() -> Dict[str, Any]:
    info: Dict[str, Any] = {}
    info["hostname"] = socket.gethostname()
    info["user"] = os.environ.get("USERNAME") or os.environ.get("USER")
    info["system_drive"] = os.environ.get("SystemDrive", "C:")
    info["platform"] = {
        "system": platform.system(),
        "release": platform.release(),
        "version": platform.version(),
        "machine": platform.machine(),
        "processor": platform.processor(),
    }

    ps = r"""
$ci = Get-ComputerInfo
[pscustomobject]@{
    WindowsProductName = $ci.WindowsProductName
    WindowsVersion      = $ci.WindowsVersion
    OsBuildNumber       = $ci.OsBuildNumber
    WindowsEditionId    = $ci.WindowsEditionId
    BiosFirmwareType    = $ci.BiosFirmwareType
    CsManufacturer      = $ci.CsManufacturer
    CsModel              = $ci.CsModel
}
"""

    data, err = ps_json(ps)
    info["windows"] = data if data is not None else {"error": err}
    return info

def collect_cpu_ram() -> Dict[str, Any]:
```

```

ps = r"""
$cpu = Get-CimInstance Win32_Processor | Select-Object Name, Manufacturer, Nu
$cs  = Get-CimInstance Win32_ComputerSystem | Select-Object TotalPhysicalMemo
[pscustomobject]@{
    CPU = $cpu
    TotalPhysicalMemory = $cs.TotalPhysicalMemory
}
"""

data, err = ps_json(ps)
if data is None:
    return {"error": err}
cpu_list = ensure_list(data.get("CPU"))
total_mem = data.get("TotalPhysicalMemory")
return {
    "cpu": cpu_list,
    "ram_bytes": total_mem,
    "ram_gb": bytes_to_gb(total_mem),
}

def collect_firmware_secureboot_tpm() -> Dict[str, Any]:
    info: Dict[str, Any] = {}

    # Firmware type: prefer Get-ComputerInfo BiosFirmwareType, fallback to regist
    fw_mode = None
    fw_err = None

    ps_fw = r"""
try {
    $ci = Get-ComputerInfo
    $ci.BiosFirmwareType
} catch {
    $null
}
"""

    code, out, err = run_powershell(ps_fw)
    if code == 0 and out:
        fw_mode = out.strip()
    else:
        ps_reg = r"""
try {
    (Get-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control' -Name
} catch { $null }
"""

```

```
""""

code2, out2, err2 = run_powershell(ps_reg)
if code2 == 0 and out2:
    try:
        v = int(out2.strip())
        fw_mode = "UEFI" if v == 2 else ("BIOS" if v == 1 else "UNKNOWN")
    except Exception:
        fw_mode = None
else:
    fw_err = err2 or err or out2 or out

info["firmware"] = {
    "boot_mode": fw_mode or "UNKNOWN",
    "error": fw_err,
}

# Secure Boot (needs UEFI + often admin)
ps_sb = r""""
try { Confirm-SecureBootUEFI } catch { "ERROR: " + $_.Exception.Message }
""""

code, out, err = run_powershell(ps_sb)
sb_val: Optional[bool] = None
sb_note: Optional[str] = None
if out:
    o = out.strip()
    if o.lower() == "true":
        sb_val = True
    elif o.lower() == "false":
        sb_val = False
    elif o.startswith("ERROR:"):
        sb_note = o
if err and not sb_note:
    sb_note = err

info["secure_boot"] = {"enabled": sb_val, "raw": out or None, "error": sb_note}

# TPM
ps_tpm = r""""
$tpm1 = $null
try { $tpm1 = Get-Tpm } catch { $tpm1 = $null }

$tpm2 = $null
try {
```

```

    $tpm2 = Get-CimInstance -Namespace root\cimv2\security\microsof்த்தpm -ClassN
        Select-Object SpecVersion, ManufacturerIdTxt, ManufacturerVersion,
    } catch { $tpm2 = $null }

[pscustomobject]@{ GetTpm = $tpm1; Win32Tpm = $tpm2 }
"""

data, err = ps_json(ps_tpm)
if data is None:
    info["tpm"] = {"error": err}
    return info

gettpm = data.get("GetTpm")
win32_list = ensure_list(data.get("Win32Tpm"))

spec_versions = []
for item in win32_list:
    sv = safe_get(item, "SpecVersion")
    if sv:
        spec_versions.append(str(sv))

tpm2_supported = None
if spec_versions:
    tpm2_supported = any(re.search(r"\b2\.0\b", sv) for sv in spec_versions)

info["tpm"] = {
    "gettpm": gettpm,
    "win32tpm": win32_list,
    "spec_versions": spec_versions,
    "tpm2_supported": tpm2_supported,
    "error": None,
}
return info

def collect_disks_partitions_volumes() -> Dict[str, Any]:
    info: Dict[str, Any] = {}

    ps_disks = r"Get-Disk | Select-Object Number, FriendlyName, SerialNumber, Bus
disks, err = ps_json(ps_disks)
disks_list = ensure_list(disks) if disks is not None else []
if disks is None:
    info["disks_error"] = err

```

```
ps_parts = r"Get-Partition | Select-Object DiskNumber, PartitionNumber, Drive
parts, err2 = ps_json(ps_parts)
parts_list = ensure_list(parts) if parts is not None else []
if parts is None:
    info["partitions_error"] = err2

ps_vols = r"Get-Volume | Select-Object DriveLetter, FileSystemLabel, FileSyst
vols, err3 = ps_json(ps_vols)
vols_list = ensure_list(vols) if vols is not None else []
if vols is None:
    info["volumes_error"] = err3

for d in disks_list:
    d["SizeGB"] = bytes_to_gb(d.get("Size"))
for p in parts_list:
    p["SizeGB"] = bytes_to_gb(p.get("Size"))
for v in vols_list:
    v["SizeGB"] = bytes_to_gb(v.get("Size"))
    v["FreeGB"] = bytes_to_gb(v.get("SizeRemaining"))

info["disks"] = disks_list
info["partitions"] = parts_list
info["volumes"] = vols_list
return info

def find_system_disk_partitionstyle(report: Dict[str, Any]) -> Tuple[Optional[int]
    system_drive = report.get("basic", {}).get("system_drive", "C:")
    drive_letter = system_drive.replace(":", "").upper()

    parts = report.get("storage", {}).get("partitions", [])
    disks = report.get("storage", {}).get("disks", [])

    disk_number = None
    for p in parts:
        dl = p.get("DriveLetter")
        if dl and str(dl).upper() == drive_letter:
            disk_number = p.get("DiskNumber")
            break

    if disk_number is None:
        for d in disks:
            if d.get("IsSystem") is True:
```

```
        disk_number = d.get("Number")
        break

partition_style = None
if disk_number is not None:
    for d in disks:
        if d.get("Number") == disk_number:
            partition_style = d.get("PartitionStyle")
            break

return disk_number, partition_style


def assess_windows11_readiness(report: Dict[str, Any]) -> Dict[str, Any]:
    checks: List[Dict[str, Any]] = []
    verdict = "UNKNOWN"

    cpu_info = report.get("hw", {}).get("cpu", [])
    addr_widths = []
    for c in cpu_info:
        aw = c.get("AddressWidth")
        if aw is not None:
            addr_widths.append(aw)
    cpu_64 = True if (addr_widths and all(int(x) >= 64 for x in addr_widths)) else

    checks.append({
        "check": "CPU 64-bit",
        "required": "Yes",
        "value": "64-bit" if cpu_64 else ("Not 64-bit" if cpu_64 is False else "U
        "pass": True if cpu_64 else False if cpu_64 is False else None,
    })

    ram_gb = report.get("hw", {}).get("ram_gb")
    ram_ok = (ram_gb >= 4.0) if ram_gb is not None else None
    checks.append({
        "check": "RAM",
        "required": ">= 4 GB",
        "value": f"{ram_gb} GB" if ram_gb is not None else "Unknown",
        "pass": ram_ok,
    })

    system_drive = report.get("basic", {}).get("system_drive", "C:")
    sys_letter = system_drive.replace(":", "").upper()
```

```
volumes = report.get("storage", {}).get("volumes", [])
sys_vol = None
for v in volumes:
    dl = v.get("DriveLetter")
    if dl and str(dl).upper() == sys_letter:
        sys_vol = v
        break

total_gb = sys_vol.get("SizeGB") if sys_vol else None
free_gb = sys_vol.get("FreeGB") if sys_vol else None

storage_ok = (total_gb >= 64.0) if total_gb is not None else None
free_ok = (free_gb >= 25.0) if free_gb is not None else None

checks.append({
    "check": "System drive capacity",
    "required": ">= 64 GB (Windows 11 requirement)",
    "value": f"{total_gb} GB" if total_gb is not None else "Unknown",
    "pass": storage_ok,
})

checks.append({
    "check": "System drive free space",
    "required": ">= 25 GB (recommended for upgrade headroom)",
    "value": f"{free_gb} GB free" if free_gb is not None else "Unknown",
    "pass": free_ok,
})

boot_mode = report.get("security", {}).get("firmware", {}).get("boot_mode")
uefi_ok = True if str(boot_mode).upper().startswith("UEFI") else (False if st
checks.append({
    "check": "Boot mode",
    "required": "UEFI",
    "value": boot_mode or "Unknown",
    "pass": uefi_ok,
})

sb = report.get("security", {}).get("secure_boot", {}).get("enabled")
sb_ok = sb if isinstance(sb, bool) else None
checks.append({
    "check": "Secure Boot",
    "required": "Enabled",
    "value": pretty_bool(sb),
```



```

    "pass": sb_ok,
    "note": report.get("security", {}).get("secure_boot", {}).get("error"),
})

tpm = report.get("security", {}).get("tpm", {})
tpm2_supported = tpm.get("tpm2_supported") if isinstance(tpm, dict) else None
gettpm = tpm.get("gettpm") if isinstance(tpm, dict) else None
tpm_present = safe_get(gettpm, "TpmPresent") if gettpm else None
tpm_ready = safe_get(gettpm, "TpmReady") if gettpm else None

tpm_ok = None
if isinstance(tpm_present, bool) and isinstance(tpm_ready, bool):
    if tpm2_supported is False:
        tpm_ok = False
    elif tpm2_supported is True:
        tpm_ok = (tpm_present and tpm_ready)
    else:
        tpm_ok = (tpm_present and tpm_ready)

checks.append({
    "check": "TPM",
    "required": "TPM 2.0 present & ready",
    "value": f"Present={pretty_bool(tpm_present)}, Ready={pretty_bool(tpm_ready)}",
    "pass": tpm_ok,
    "note": tpm.get("error") if isinstance(tpm, dict) else None,
})

disk_no, part_style = find_system_disk_partitionstyle(report)
gpt_ok = None
if part_style:
    gpt_ok = True if str(part_style).upper() == "GPT" else False if str(part_

checks.append({
    "check": "System disk partition style",
    "required": "GPT (recommended/typical for UEFI + Win11)",
    "value": f"Disk {disk_no}: {part_style}" if disk_no is not None else (par
    "pass": gpt_ok,
})

key_passes = [c["pass"] for c in checks if c["pass"] is not None]
if any(p is False for p in key_passes):
    verdict = "NOT READY (some requirements failed)"
elif key_passes and all(p is True for p in key_passes):

```

```
        verdict = "READY (requirements look OK)"
    else:
        verdict = "PARTIAL/UNKNOWN (some checks couldn't be confirmed)"

    actions: List[str] = []
    if free_ok is False:
        actions.append("Free up space on C: aim for 25-30 GB free before upgrade.")
    if uefi_ok is False:
        actions.append("Switch boot mode to UEFI in BIOS/UEFI settings.")
    if gpt_ok is False:
        actions.append("System disk is MBR → consider converting to GPT (mbr2gpt.")
    if sb_ok is False:
        actions.append("Enable Secure Boot in UEFI settings.")
    if tpm_ok is False:
        actions.append("Enable TPM 2.0 in BIOS/UEFI (Intel PTT / AMD fTPM).")

    return {"verdict": verdict, "checks": checks, "suggested_actions": actions}

def render_text_report(report: Dict[str, Any]) -> str:
    lines: List[str] = []
    basic = report.get("basic", {})
    win = basic.get("windows", {})
    hw = report.get("hw", {})
    sec = report.get("security", {})
    storage = report.get("storage", {})
    assess = report.get("assessment", {})

    lines.append("=== Windows 11 Readiness / System Inventory Report (v2) ===")
    lines.append(f"Generated: {report.get('generated_at')}")
    lines.append(f"Host: {basic.get('hostname')} | User: {basic.get('user')} | Sy")
    lines.append("")

    lines.append("--- Windows ---")
    if isinstance(win, dict) and "error" not in win:
        lines.append(f"Product: {win.get('WindowsProductName')}")
        lines.append(f"Version: {win.get('WindowsVersion')} | Build: {win.get('Os")
        lines.append(f"Firmware (from ComputerInfo): {win.get('BiosFirmwareType')}")
        lines.append(f"Manufacturer/Model: {win.get('CsManufacturer')} {win.get('")
    else:
        lines.append(f"Windows info error: {win}")
    lines.append("")
```

```
lines.append("--- CPU / RAM ---")
cpu_list = hw.get("cpu", [])
if cpu_list and isinstance(cpu_list, list):
    for i, c in enumerate(cpu_list, start=1):
        lines.append(f"CPU #{i}: {c.get('Name')} | Cores: {c.get('NumberOfCores')}")
else:
    lines.append(f"CPU info: {hw}")
lines.append(f"RAM: {hw.get('ram_gb')} GB")
lines.append("")

lines.append("--- Firmware / Security ---")
fw = sec.get("firmware", {})
lines.append(f"Boot mode: {fw.get('boot_mode')} | note: {fw.get('error')}")
sb = sec.get("secure_boot", {})
lines.append(f"Secure Boot: {pretty_bool(sb.get('enabled'))} | note: {sb.get('error')}")
tpm = sec.get("tpm", {})
if isinstance(tpm, dict) and "error" not in tpm:
    gettpm = tpm.get("gettpm") or {}
    lines.append(
        "TPM: "
        f"Present={pretty_bool(safe_get(gettpm, 'TpmPresent'))}, "
        f"Ready={pretty_bool(safe_get(gettpm, 'TpmReady'))}, "
        f"TPM2={pretty_bool(tpm.get('tpm2_supported'))}, "
        f"SpecVersions={tpm.get('spec_versions')}"
    )
else:
    lines.append(f"TPM info: {tpm}")
lines.append("")

lines.append("--- Disks ---")
for d in storage.get("disks", []):
    lines.append(
        f"Disk {d.get('Number')}: {d.get('FriendlyName')} | {d.get('SizeGB')} GB "
        f"Style={d.get('PartitionStyle')} | Bus={d.get('BusType')} | "
        f"IsSystem={d.get('IsSystem')} IsBoot={d.get('IsBoot')}"
    )
lines.append("")

lines.append("--- Volumes ---")
for v in storage.get("volumes", []):
    dl = v.get("DriveLetter")
    if not dl:
        continue
```

```

        lines.append(
            f"{dl}: | FS={v.get('FileSystemType')} | Label={v.get('FileSystemLabel')} | Size={v.get('SizeGB')} GB | Free={v.get('FreeGB')} GB | Health={v.get('Health')}
        )
    lines.append("")

    lines.append("=== Windows 11 Readiness Checks ===")
    lines.append(f"Verdict: {assess.get('verdict')}")
    lines.append("")
    for c in assess.get("checks", []):
        status = "PASS" if c["pass"] is True else ("FAIL" if c["pass"] is False else "UNKNOWN")
        note = f" | note: {c.get('note')}" if c.get("note") else ""
        lines.append(f"- [{status}] {c['check']}: {c['value']} (required: {c['required']})")

    actions = assess.get("suggested_actions", [])
    if actions:
        lines.append("")
        lines.append("Next steps (based on blockers):")
        for a in actions:
            lines.append(f"  * {a}")

    return "\n".join(lines)

def main():
    if platform.system().lower() != "windows":
        print("This script is intended for Windows 10/11 only.")
        sys.exit(1)

    ap = argparse.ArgumentParser(description="Collect system/disk/security info and generate a report")
    ap.add_argument("--outdir", default=".", help="Directory to save report files")
    args = ap.parse_args()

    outdir = os.path.abspath(args.outdir)
    os.makedirs(outdir, exist_ok=True)

    report: Dict[str, Any] = {
        "generated_at": datetime.now().isoformat(timespec="seconds"),
        "basic": collect_basic_system(),
        "hw": collect_cpu_ram(),
        "security": collect_firmware_secureboot_tpm(),
        "storage": collect_disks_partitions_volumes(),
    }

```

```
report["assessment"] = assess_windows11_readiness(report)

text = render_text_report(report)

host = report["basic"].get("hostname", "host")
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base_name = f"win11_readiness_v2_{host}_{ts}"

json_path = os.path.join(outdir, base_name + ".json")
txt_path = os.path.join(outdir, base_name + ".txt")

with open(json_path, "w", encoding="utf-8") as f:
    json.dump(report, f, ensure_ascii=False, indent=2)
with open(txt_path, "w", encoding="utf-8") as f:
    f.write(text)

print(text)
print(f"\nSaved:\n- {txt_path}\n- {json_path}")

if __name__ == "__main__":
    main()
```

Запуск (лучше так)

1. Открой PowerShell от имени администратора
2. Перейди на Desktop:

```
powershell
```

 Копировать код

```
cd $env:USERPROFILE\Desktop
```

3. Запусти:

```
powershell
```

 Копировать код

```
python .\win11_readiness_report_v2.py --outdir .
```

3) Что делать прямо сейчас, чтобы Win11 вообще согласилась ставиться

А) Срочно освободить место на C: до 25–30 GB (у тебя 6.7 GB)

Самые быстрые “хирургические” ходы (по убыванию профита):

1. Выключить гибернацию (часто освобождает несколько ГБ мгновенно):

```
powershell
```

 Копировать код

```
powercfg -h off
```

2. Очистка компонентов Windows (WinSxS):

```
powershell
```

 Копировать код

```
DISM /Online /Cleanup-Image /StartComponentCleanup
```

3. Disk Cleanup / Storage Sense

- Параметры → Система → Память → Временные файлы → удалить.
- Или:

```
powershell
```

 Копировать код

```
cleanmgr
```

4. Перенести жирное с C на D (у тебя D свободно ~106 GB — прям курорт)

- Перенести: Downloads , Desktop , папки с играми/архивами, кэш.
- Для “Документы/Загрузки/Рабочий стол”: ПКМ по папке → **Свойства** → **Расположение** → **Переместить...** на D.

5. Если есть windows.old (после обновлений) — удалить через “Временные файлы”.

Если хочешь, могу дать мини-скрипт, который **топ-20 самых жирных папок на C:** покажет (без сторонних либ).

B) Secure Boot/TPM/UEFI

У тебя сейчас Confirm-SecureBootUEFI ругается на **PermissionDenied** — это почти всегда потому что PowerShell был не админ.

Запусти v2 **от админа**, и станет понятно:

- Boot mode: UEFI или BIOS
- TPM: Present/Ready + TPM2
- Secure Boot: True/False

4) По твоему текущему состоянию “как есть”

Если после запуска v2 окажется, что:

- UEFI = да
- TPM2 = да
- Secure Boot = можно включить
- С свободно ≥ 25 GB

— тогда апгрейд до Win11 будет обычной процедурой, без шаманства.

Сейчас единственный железный стоппер, который уже виден — **С: мало места.**

Сделай так: запусти **v2** от админа и кинь сюда только блок **"Firmware/Security + CPU/RAM + Verdict"** — скажу точно, что у тебя по TPM/SecureBoot/UEFI и нужно ли лезть в BIOS или можно просто "похудеть С:" и обновляться.



```
PS C:\Users\Славиус> & C:/Users/
Славиус/AppData/Local/Programs/Python/Python313/pyt
hon.exe c:/Users/
Славиус/Desktop/win11_readiness_report_v2.py
=== Windows 11 Readiness / System Inventory Report
(v2) ===
Generated: 2025-12-27T14:16:16
Host: DESKTOP-1OAT63D | User: Славиус | SystemDrive: C:
```

--- Windows ---

Product: Windows 10 Home
Version: 2009 | Build: 19045 | Edition: Core
Firmware (from ComputerInfo): 2
Manufacturer/Model: MSI MS-7817

--- CPU / RAM ---

CPU #1: Intel(R) Core(TM) i3-4170 CPU @ 3.70GHz | Cores:
2 | Threads: 4 | AddrWidth: 64
RAM: 11.95 GB

--- Firmware / Security ---

Boot mode: Uefi
Secure Boot: UNKNOWN | note: ERROR: Unable to set
proper privileges. Access was denied.
TPM info: {'gettpm': 'Administrator privilege is required to

```
execute this command.', 'win32tpm': [{}], 'spec_versions': [],  
'tpm2_supported': None, 'error': None}
```

--- Disks ---

Disk 0: P4-240 | 223.57 GB | Style=GPT | Bus=SATA |
IsSystem=True IsBoot=True
Disk 1: SAMSUNG HD161GJ | 149.05 GB | Style=MBR |
Bus=SATA | IsSystem=False IsBoot=False

--- Volumes ---

G: | FS=NTFS | Label= | Size=24.04 GB | Free=17.57 GB |
Health=Healthy
E: | FS=NTFS | Label=Win 7 | Size=30.01 GB | Free=29.93
GB | Health=Healthy
F: | FS=NTFS | Label= | Size=95.0 GB | Free=49.87 GB |
Health=Healthy
C: | FS=NTFS | Label= | Size=111.4 GB | Free=6.78 GB |
Health=Healthy
D: | FS=NTFS | Label=Новый том | Size=111.4 GB |
Free=106.38 GB | Health=Healthy
H: | FS=Unknown | Label= | Size=0.0 GB | Free=0.0 GB |
Health=Healthy

=== Windows 11 Readiness Checks ===

Verdict: NOT READY (some requirements failed)

- [PASS] CPU 64-bit: 64-bit (required: Yes)
- [PASS] RAM: 11.95 GB (required: >= 4 GB)
- [PASS] System drive capacity: 111.4 GB (required: >= 64 GB (Windows 11 requirement))
- [FAIL] System drive free space: 6.78 GB free (required: >= 25 GB (recommended for upgrade headroom))
- [PASS] Boot mode: Uefi (required: UEFI)
- [UNKNOWN] Secure Boot: UNKNOWN (required: Enabled) | note: ERROR: Unable to set proper privileges. Access was denied.
- [UNKNOWN] TPM: Present=UNKNOWN, Ready=UNKNOWN, TPM2=UNKNOWN (required: TPM 2.0 present & ready)
- [PASS] System disk partition style: Disk 0: GPT (required: GPT (recommended/typical for UEFI + Win11))

Next steps (based on blockers):

* Free up space on C: aim for 25–30 GB free before upgrade.

Saved:

- C:\Users\Славиус\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141638.txt

- C:\Users\Славиус\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141638.json

PS C:\Users\Славиус>

повершел запуск а не первый через windsurf:

Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

PS C:\Windows\system32> cd \$env:USERPROFILE\Desktop

>> python .\win11_readiness_report_v2.py --outdir .

>>

=== Windows 11 Readiness / System Inventory Report (v2) ===

Generated: 2025-12-27T14:15:56

Host: DESKTOP-1OAT63D | User: Славиус | SystemDrive: C:

--- Windows ---

Product: Windows 10 Home

Version: 2009 | Build: 19045 | Edition: Core

Firmware (from ComputerInfo): 2

Manufacturer/Model: MSI MS-7817

--- CPU / RAM ---

CPU #1: Intel(R) Core(TM) i3-4170 CPU @ 3.70GHz | Cores:

2 | Threads: 4 | AddrWidth: 64

RAM: 11.95 GB

--- Firmware / Security ---

Boot mode: Uefi

Secure Boot: NO

TPM info: {'gettpm': {'TpmPresent': False, 'TpmReady': False, 'TpmEnabled': False, 'TpmActivated': False, 'TpmOwned': False, 'RestartPending': False, 'ManufacturerId': 0,

```
'ManufacturerIdTxt': None, 'ManufacturerVersion': None,  
'ManufacturerVersionFull20': None, 'ManagedAuthLevel': 4,  
'OwnerAuth': None, 'OwnerClearDisabled': True,  
'AutoProvisioning': -1, 'LockedOut': False,  
'LockoutHealTime': None, 'LockoutCount': None,  
'LockoutMax': None, 'SelfTest': None}, 'win32tpm': [{}],  
'spec_versions': [], 'tpm2_supported': None, 'error': None}
```

--- Disks ---

```
Disk 0: P4-240 | 223.57 GB | Style=GPT | Bus=SATA |  
IsSystem=True IsBoot=True  
Disk 1: SAMSUNG HD161GJ | 149.05 GB | Style=MBR |  
Bus=SATA | IsSystem=False IsBoot=False
```

--- Volumes ---

```
G: | FS=NTFS | Label= | Size=24.04 GB | Free=17.57 GB |  
Health=Healthy  
E: | FS=NTFS | Label=Win 7 | Size=30.01 GB | Free=29.93  
GB | Health=Healthy  
F: | FS=NTFS | Label= | Size=95.0 GB | Free=49.87 GB |  
Health=Healthy  
C: | FS=NTFS | Label= | Size=111.4 GB | Free=6.78 GB |  
Health=Healthy  
D: | FS=NTFS | Label=Новый том | Size=111.4 GB |  
Free=106.38 GB | Health=Healthy  
H: | FS=Unknown | Label= | Size=0.0 GB | Free=0.0 GB |  
Health=Healthy
```

=== Windows 11 Readiness Checks ===

Verdict: NOT READY (some requirements failed)

- [PASS] CPU 64-bit: 64-bit (required: Yes)
- [PASS] RAM: 11.95 GB (required: >= 4 GB)
- [PASS] System drive capacity: 111.4 GB (required: >= 64 GB (Windows 11 requirement))
- [FAIL] System drive free space: 6.78 GB free (required: >= 25 GB (recommended for upgrade headroom))
- [PASS] Boot mode: Uefi (required: UEFI)
- [FAIL] Secure Boot: NO (required: Enabled)
- [FAIL] TPM: Present=NO, Ready=NO, TPM2=UNKNOWN (required: TPM 2.0 present & ready)
- [PASS] System disk partition style: Disk 0: GPT (required: GPT (recommended/typical for UEFI + Win11))

Next steps (based on blockers):






- * Free up space on C: aim for 25–30 GB free before upgrade.
- * Enable Secure Boot in UEFI settings.
- * Enable TPM 2.0 in BIOS/UEFI (Intel PTT / AMD fTPM).

Saved:

```
-
C:\Users\Славиус\Desktop\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141624.txt
-
C:\Users\Славиус\Desktop\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141624.json
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop>
документы перенести не удалось
тк какая то папка мешает
ида
напиши скрипт для места
```

Думал на протяжении 1m 1s >

Отчёт v2 уже прям честный, как бухгалтер после двух кофе:

-  UEFI = да, GPT на системном диске = да, CPU 64-bit = да, RAM = 12GB — норм
-  C: свободно 6.78 GB — это главный "NOPE" для апгрейда
-  Secure Boot = NO
-  TPM = TpmPresent False (то есть в системе его реально нет/выключен/не поддерживается)
-  И ещё важное: Intel i3-4170 (4-е поколение) — официально не поддерживается Win11 (даже если включить SecureBoot/TPM). Поставить можно обходными путями, но Microsoft это не гарантирует.

Ты попросил: "напиши скрипт для места" — держи. Он покажет:

- сколько свободно на всех дисках,
- топ папок по размеру (на уровне 1 и 2),
- топ самых жирных файлов (hiberfil/pagefile/iso/архивы),

- сохранит отчёт в TXT+CSV+JSON.

Скрипт: diskspace_usage_win.py (без сторонних библиотек)

Сохрани на Desktop как diskspace_usage_win.py и запускай.

python

 Копировать код

```
# diskspace_usage_win.py
# Windows disk usage analyzer: top folders + top files, with safe scanning (no sy
# Python 3.9+ (works on 3.13).
#
# Examples:
# python .\diskspace_usage_win.py
# python .\diskspace_usage_win.py --root C:\ --depth 2 --top 30 --files 50
# python .\diskspace_usage_win.py --root "C:\Users\Славиус" --depth 3
# python .\diskspace_usage_win.py --roots "C:\Users\Славиус" "C:\Windows\Softwa
#
# Tip: run PowerShell as Administrator for fewer "Access denied".
```

```
from __future__ import annotations
```

```
import argparse
import csv
import heapq
import json
import os
import shutil
import sys
import time
from dataclasses import dataclass, asdict
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Tuple, Optional
```

```
REPARSE_POINT_ATTR = 0x0400 # FILE_ATTRIBUTE_REPARSE_POINT
```

```
def is_reparse_point(path: str) -> bool:
    try:
        st = os.stat(path, follow_symlinks=False)
```

```
        attrs = getattr(st, "st_file_attributes", 0)
        return bool(attrs & REPARSE_POINT_ATTR)
    except Exception:
        return False

def format_gb(num_bytes: float) -> str:
    return f"{num_bytes / (1024**3):.2f} GB"

def list_drives() -> List[str]:
    drives = []
    for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        p = f"{c}:\\"
        if os.path.exists(p):
            drives.append(p)
    return drives

@dataclass
class ScanStats:
    root: str
    depth: int
    files_scanned: int = 0
    dirs_scanned: int = 0
    bytes_total: int = 0
    errors: int = 0
    denied: int = 0
    skipped_reparse: int = 0
    elapsed_sec: float = 0.0

@dataclass
class ScanResult:
    stats: ScanStats
    top_dirs_level1: List[Tuple[str, int]]
    top_dirs_level2: List[Tuple[str, int]]
    top_files: List[Tuple[str, int]]

def scan_root(root: str, depth: int, top_dirs: int, top_files: int) -> ScanResult
    """
    Walk root without following symlinks/reparse points.
```

```
Aggregate sizes for level1 and level2 dirs relative to root.
"""

t0 = time.time()
root = os.path.abspath(root)
stats = ScanStats(root=root, depth=depth)

# Aggregations
lvl1: Dict[str, int] = {}
lvl2: Dict[str, int] = {}

# Keep a min-heap for top files (size, path)
file_heap: List[Tuple[int, str]] = []

# Stack DFS: (dirpath)
stack = [root]

root_parts_len = len(Path(root).parts)

def add_to_heap(sz: int, p: str):
    if top_files <= 0:
        return
    if len(file_heap) < top_files:
        heapq.heappush(file_heap, (sz, p))
    else:
        if sz > file_heap[0][0]:
            heapq.heapreplace(file_heap, (sz, p))

while stack:
    cur = stack.pop()
    stats.dirs_scanned += 1

    try:
        with os.scandir(cur) as it:
            for entry in it:
                try:
                    # Skip symlinks / reparse points to avoid loops
                    if entry.is_symlink():
                        continue
                    full_path = entry.path

                    # Skip reparse points (junctions, OneDrive placeholders,
                    if is_reparse_point(full_path):
                        stats.skipped_reparse += 1
```

```
        continue

    if entry.is_dir(follow_symlinks=False):
        # Optional depth limit (depth=0 -> only root listing)
        if depth >= 0:
            # compute relative depth
            rel_parts = len(Path(full_path).parts) - root_parts
            if rel_parts <= depth:
                stack.append(full_path)
        else:
            stack.append(full_path)

    elif entry.is_file(follow_symlinks=False):
        st = entry.stat(follow_symlinks=False)
        sz = int(getattr(st, "st_size", 0))
        stats.files_scanned += 1
        stats.bytes_total += sz

        # Determine relative grouping for lvl1 / lvl2
        try:
            rel = os.path.relpath(full_path, root)
        except Exception:
            rel = full_path

        parts = rel.split(os.sep)
        if len(parts) >= 1:
            key1 = parts[0]
            lvl1[key1] = lvl1.get(key1, 0) + sz
        if len(parts) >= 2:
            key2 = os.path.join(parts[0], parts[1])
            lvl2[key2] = lvl2.get(key2, 0) + sz

        add_to_heap(sz, full_path)

except PermissionError:
    stats.denied += 1
except FileNotFoundError:
    # race condition: file disappeared while scanning
    stats.errors += 1
except OSError:
    stats.errors += 1
```

```
except PermissionError:
```

```

        stats.denied += 1
    except FileNotFoundError:
        stats.errors += 1
    except OSError:
        stats.errors += 1

stats.elapsed_sec = round(time.time() - t0, 2)

# Sort top dirs
top1 = sorted(lvl1.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top2 = sorted(lvl2.items(), key=lambda x: x[1], reverse=True)[:top_dirs]

# Extract heap into sorted list
top_files_list = sorted([(p, sz) for (sz, p) in file_heap], key=lambda x: x[1]

return ScanResult(
    stats=stats,
    top_dirs_level1=top1,
    top_dirs_level2=top2,
    top_files=top_files_list,
)

def special_c_hogs(system_drive: str = "C:\\") -> List[Tuple[str, Optional[int]]]
"""
Check sizes of well-known huge files/folders on system drive.
"""
candidates = [
    os.path.join(system_drive, "hiberfil.sys"),
    os.path.join(system_drive, "pagefile.sys"),
    os.path.join(system_drive, "swapfile.sys"),
    os.path.join(system_drive, "Windows.old"),
    os.path.join(system_drive, "Windows", "SoftwareDistribution", "Download"),
    os.path.join(system_drive, "Windows", "Temp"),
    os.path.join(system_drive, "Users"),
    os.path.join(system_drive, "$Recycle.Bin"),
]

out = []
for p in candidates:
    try:
        if os.path.isfile(p):
            out.append((p, os.path.getsize(p)))

```



```

        elif os.path.isdir(p):
            # folder size can be huge to compute; mark as None (use scan to m
            out.append((p, None))
        else:
            out.append((p, None))
    except Exception:
        out.append((p, None))
return out

def print_drive_table():
    print("=== Drives ===")
    for d in list_drives():
        try:
            du = shutil.disk_usage(d)
            used = du.total - du.free
            print(f"{d} Total: {format_gb(du.total)} | Used: {format_gb(used)} |")
        except Exception as e:
            print(f"{d} (error reading usage: {e})")
    print()

def main():
    if os.name != "nt":
        print("This script is intended for Windows.")
        sys.exit(1)

    ap = argparse.ArgumentParser(description="Disk space analyzer for Windows (to
    ap.add_argument("--root", default="", help="Single root to scan (e.g. C:\\ or
    ap.add_argument("--roots", nargs="*", default=[], help="Multiple roots to sca
    ap.add_argument("--depth", type=int, default=6, help="Max relative depth to s
    ap.add_argument("--top", type=int, default=25, help="Top N directories to sho
    ap.add_argument("--files", type=int, default=30, help="Top N files to show (d
    ap.add_argument("--outdir", default=".", help="Where to save TXT/CSV/JSON rep
    args = ap.parse_args()

    outdir = os.path.abspath(args.outdir)
    os.makedirs(outdir, exist_ok=True)

    print_drive_table()

    system_drive = os.environ.get("SystemDrive", "C:") + "\\\"
    userprofile = os.environ.get("USERPROFILE", "")

```

```
# Default scan targets if none provided
roots: List[str] = []
if args.root:
    roots.append(args.root)
roots.extend(args.roots)

if not roots:
    # practical defaults: user profile + Windows download cache
    roots = [
        userprofile or os.path.join(system_drive, "Users"),
        os.path.join(system_drive, "Windows", "SoftwareDistribution", "Downlo
        os.path.join(system_drive, "Windows", "Temp"),
    ]

# Print known hogs
print("=== Quick suspects on C: ===")
for p, sz in special_c_hogs(system_drive):
    if sz is None:
        print(f"- {p} (folder/unknown size – scan to measure)")
    else:
        print(f"- {p} {format_gb(sz)}")
print()

all_results: List[ScanResult] = []

for r in roots:
    if not r:
        continue
    if not os.path.exists(r):
        print(f"[skip] Root not found: {r}")
        continue

    print(f"=== Scanning: {r} (depth={args.depth}) ===")
    res = scan_root(r, depth=args.depth, top_dirs=args.top, top_files=args.fi
    all_results.append(res)

    st = res.stats
    print(f"Scanned dirs={st.dirs_scanned}, files={st.files_scanned}, total={
        f"denied={st.denied}, errors={st.errors}, skipped_reparse={st.skipp
    print()

    print("--- Top folders (level 1) ---")
```

```
for name, sz in res.top_dirs_level1:
    print(f"{name:<45} {format_gb(sz)}")
print()

print("--- Top folders (level 2) ---")
for name, sz in res.top_dirs_level2:
    print(f"{name:<45} {format_gb(sz)}")
print()

print("--- Top files ---")
for path, sz in res.top_files:
    print(f"{format_gb(sz):>10} {path}")
print("\n" + "=" * 70 + "\n")

# Save reports
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base = f"diskspace_report_{ts}"
txt_path = os.path.join(outdir, base + ".txt")
json_path = os.path.join(outdir, base + ".json")
csv_path = os.path.join(outdir, base + "_topdirs.csv")

# TXT
with open(txt_path, "w", encoding="utf-8") as f:
    f.write(f"Disk space report generated: {datetime.now().isoformat(timespec='seconds')}")
    f.write("Drives:\n")
    for d in list_drives():
        try:
            du = shutil.disk_usage(d)
            used = du.total - du.free
            f.write(f"{d} Total {du.total} Used {used} Free {du.free}\n")
        except Exception as e:
            f.write(f"{d} error {e}\n")
    f.write("\n")

for res in all_results:
    st = res.stats
    f.write(f"Scan root: {st.root}\n")
    f.write(f"dirs={st.dirs_scanned} files={st.files_scanned} bytes_total={st.bytes_total}\n")
    f.write(f"denied={st.denied} errors={st.errors} skipped_reparse={st.skipped_reparse}\n")

    f.write("Top folders level 1:\n")
    for name, sz in res.top_dirs_level1:
        f.write(f"{name}\t{sz}\n")
```

```
f.write("\nTop folders level 2:\n")
for name, sz in res.top_dirs_level2:
    f.write(f"{name}\t{sz}\n")
f.write("\nTop files:\n")
for path, sz in res.top_files:
    f.write(f"{path}\t{sz}\n")
f.write("\n" + "-" * 60 + "\n\n")

# JSON
with open(json_path, "w", encoding="utf-8") as f:
    payload = {
        "generated_at": datetime.now().isoformat(timespec="seconds"),
        "results": [
            {
                "stats": asdict(res.stats),
                "top_dirs_level1": res.top_dirs_level1,
                "top_dirs_level2": res.top_dirs_level2,
                "top_files": res.top_files,
            }
            for res in all_results
        ],
    }
    json.dump(payload, f, ensure_ascii=False, indent=2)

# CSV (top level 1+2)
with open(csv_path, "w", encoding="utf-8-sig", newline="") as f:
    w = csv.writer(f, delimiter=";")
    w.writerow(["root", "level", "path", "bytes", "gb"])
    for res in all_results:
        root = res.stats.root
        for p, sz in res.top_dirs_level1:
            w.writerow([root, 1, p, sz, round(sz / (1024**3), 3)])
        for p, sz in res.top_dirs_level2:
            w.writerow([root, 2, p, sz, round(sz / (1024**3), 3)])

print(f"Saved reports:\n- {txt_path}\n- {json_path}\n- {csv_path}")

if __name__ == "__main__":
    main()
```

Как запускать (рекомендую вот так)

Открой PowerShell (лучше **Администратор**), и:

```
powershell
```

[Копировать код](#)

```
cd $env:USERPROFILE\Desktop  
python .\diskspace_usage_win.py
```

Если хочешь прицельно по "Документы":

```
powershell
```

[Копировать код](#)

```
python .\diskspace_usage_win.py --root "$env:USERPROFILE\Documents" --depth 8 --t
```

Или по всей C: (это дольше):

```
powershell
```

[Копировать код](#)

```
python .\diskspace_usage_win.py --root C:\ --depth 6 --top 40 --files 60
```

Почему "Документы не перенеслись" и "какая-то папка мешает"

Типовые убийцы:

1. **OneDrive** включил "резервное копирование" Desktop/Documents — тогда перенос через "Расположение" часто ломается.
Решение: в OneDrive → Settings → Sync and backup → **Manage backup** → отключить Documents/Desktop (временно).
2. Внутри Documents есть папка с особыми правами/битой точкой восстановления/**reparse point** (junction).
Скрипт выше как раз покажет, какая папка жирная/странная.
3. Файлы "заняты" (Word/Excel/браузер/антивирус держит). Тогда проще переносить **вручную** (копия → проверка → потом чистка).

Самая быстрая победа по месту (без философии)

Тебе надо минимум **+20 GB** на C:.

1. В админ PowerShell:

```
powershell
```

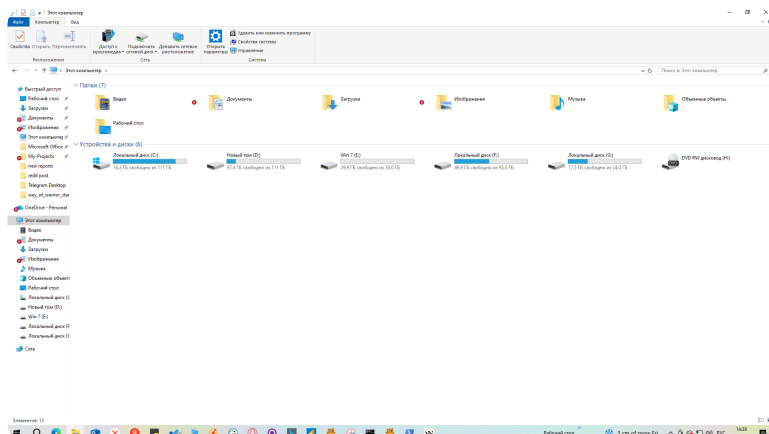
[Копировать код](#)

```
powercfg -h off
DISM /Online /Cleanup-Image /StartComponentCleanup
cleanmgr
```

2. Перенеси жирное на D: (у тебя там 106 GB свободно, это как шведский стол):

- Downloads, Games, ISO, архивы, видео
- AppData трогать аккуратно (лучше по результатам скрипта)

Если скинешь сюда вывод скрипта **Top folders level 1** для C:\Users\Славиус и для C:\ (или хотя бы скрин/кусоч), я скажу прям по пунктам: **что удалить, что переносить, что лучше не трогать**, чтобы винда не умерла в истерике.



уже вот кстати
освободил переносом загрузок
и кстати
как сохранить всю информацию от истории и паролей
до условно сессий загрузок и тд от браузеров?
yandex, opera, edge
и еще кстати сделал случайно (командой клин, потом
выяснил через другой чат какие то команды чтобы ще
кое что освободить)
вот все что было в повершел
dows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все
права защищены.

Попробуйте новую кроссплатформенную оболочку

PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Windows\system32> cd $env:USERPROFILE\Desktop  
>> python .\win11_readiness_report_v2.py --outdir .  
>>
```

=== Windows 11 Readiness / System Inventory Report
(v2) ===

Generated: 2025-12-27T14:15:56

Host: DESKTOP-1OAT63D | User: Славиус | SystemDrive: C:

--- Windows ---

Product: Windows 10 Home

Version: 2009 | Build: 19045 | Edition: Core

Firmware (from ComputerInfo): 2

Manufacturer/Model: MSI MS-7817

--- CPU / RAM ---

CPU #1: Intel(R) Core(TM) i3-4170 CPU @ 3.70GHz | Cores:
2 | Threads: 4 | AddrWidth: 64

RAM: 11.95 GB

--- Firmware / Security ---

Boot mode: Uefi

Secure Boot: NO

TPM info: {'gettpm': {'TpmPresent': False, 'TpmReady': False,
'TpmEnabled': False, 'TpmActivated': False, 'TpmOwned':
False, 'RestartPending': False, 'ManufacturerId': 0,
'ManufacturerIdTxt': None, 'ManufacturerVersion': None,
'ManufacturerVersionFull20': None, 'ManagedAuthLevel': 4,
'OwnerAuth': None, 'OwnerClearDisabled': True,
'AutoProvisioning': -1, 'LockedOut': False,
'LockoutHealTime': None, 'LockoutCount': None,
'LockoutMax': None, 'SelfTest': None}, 'win32tpm': [{}],
'spec_versions': [], 'tpm2_supported': None, 'error': None}

--- Disks ---

Disk 0: P4-240 | 223.57 GB | Style=GPT | Bus=SATA |
IsSystem=True IsBoot=True

Disk 1: SAMSUNG HD161GJ | 149.05 GB | Style=MBR |
Bus=SATA | IsSystem=False IsBoot=False

--- Volumes ---

G: | FS=NTFS | Label= | Size=24.04 GB | Free=17.57 GB |

Health=Healthy
E: | FS=NTFS | Label=Win 7 | Size=30.01 GB | Free=29.93 GB | Health=Healthy
F: | FS=NTFS | Label= | Size=95.0 GB | Free=49.87 GB | Health=Healthy
C: | FS=NTFS | Label= | Size=111.4 GB | Free=6.78 GB | Health=Healthy
D: | FS=NTFS | Label=Новый том | Size=111.4 GB | Free=106.38 GB | Health=Healthy
H: | FS=Unknown | Label= | Size=0.0 GB | Free=0.0 GB | Health=Healthy

=== Windows 11 Readiness Checks ===

Verdict: NOT READY (some requirements failed)

- [PASS] CPU 64-bit: 64-bit (required: Yes)
- [PASS] RAM: 11.95 GB (required: ≥ 4 GB)
- [PASS] System drive capacity: 111.4 GB (required: ≥ 64 GB (Windows 11 requirement))
- [FAIL] System drive free space: 6.78 GB free (required: ≥ 25 GB (recommended for upgrade headroom))
- [PASS] Boot mode: Uefi (required: UEFI)
- [FAIL] Secure Boot: NO (required: Enabled)
- [FAIL] TPM: Present=NO, Ready=NO, TPM2=UNKNOWN (required: TPM 2.0 present & ready)
- [PASS] System disk partition style: Disk 0: GPT (required: GPT (recommended/typical for UEFI + Win11))

Next steps (based on blockers):

- * Free up space on C: aim for 25–30 GB free before upgrade.
- * Enable Secure Boot in UEFI settings.
- * Enable TPM 2.0 in BIOS/UEFI (Intel PTT / AMD fTPM).

Saved:

-

C:\Users\Славиус\Desktop\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141624.txt

-

C:\Users\Славиус\Desktop\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141624.json

PS C:\Users\Славиус\Desktop>

PS C:\Users\Славиус\Desktop>


```
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop> powercfg -h off
>> DISM /Online /Cleanup-Image
/StartComponentCleanup
>> cleanmgr
>> clean
```

Deployment Image Servicing and Management tool
Version: 10.0.19041.3636

Image Version: 10.0.19045.6456

```
[=====          10.0%          ]
[=====100.0%=====]
=====]
```

The operation completed successfully.

clean : Имя "clean" не распознано как имя командлета,
функции, файла сценария или выполняемой
программы. Проверьте

правильность написания имени, а также наличие и
правильность пути, после чего повторите попытку.

строка:4 знак:1

+ clean

+ ~~~~~

+ CategoryInfo : ObjectNotFound: (clean:String) [],
CommandNotFoundException

+ FullyQualifiedErrorId : CommandNotFoundException

```
PS C:\Users\Славиус\Desktop> net stop wuauserv
>> net stop bits
>> Remove-Item -Recurse -Force
C:\Windows\SoftwareDistribution\Download\*
>> net start bits
>> net start wuauserv
>> Clear-RecycleBin -Force
>> Remove-Item -Recurse -Force $env:TEMP\*
>>
```

Думал на протяжении 2m 15s >

О, красавчик — по скрину **C:** уже **~16.3 GB свободно**. Было 6–7, стало 16 — винда хотя бы перестала задыхаться. Но для апгрейда/манёвров всё равно цель **25–30 GB** (ещё +10–15 GB).

Теперь по твоим вопросам.

1) Что ты “случайно” почистил — это норм?

Твои команды в целом **безопасные**:

- `powercfg -h off` — отключил гибернацию → **освободил hiberfil.sys** (часто 4–12 GB).
(Если захочешь обратно: `powercfg -h on .`)
- `DISM ... StartComponentCleanup` — чистит старые компоненты обновлений WinSxS → **норм**, это штатно.
- `cleanmgr` — стандартная очистка.
- `net stop wuauserv / bits` + удаление
`C:\Windows\SoftwareDistribution\Download*` — **сброс кэша обновлений**. Ничего личного не удаляет, просто Windows Update потом заново докачает.
- `Clear-RecycleBin` — очистка корзины.
- `Remove-Item $env:TEMP*` — чистка временных файлов. Иногда может “вылогинить” какие-то мелкие приложения/инсталляторы, но критично — нет.
- `clean` — это просто команда, которой **не существует**, так что ничего не сломал.

Так что тут без паники: ты выкинул мусор, а не мозг системы.

2) Как сохранить историю/пароли/сессии/загрузки браузеров (Yandex/Opera/Edge)

Есть 3 уровня “сохранения”, от самого надёжного к самому “хардкорному”.

Уровень А — самое надёжное: включить синхронизацию

Это лучший способ, если ты **меняешь ПК/делаешь чистую установку**.

- **Edge** → войти в Microsoft аккаунт → включить Sync (пароли/избранное/расширения/настройки).
- **Yandex** → войти в Яндекс → синхронизация.
- **Opera** → Opera Sync.

⚠ Но: “сессии сайтов” (куки/логины) и “список загрузок” синкают не всегда.

Поэтому...

Уровень В — экспорт: пароли/закладки (на случай если куки умрут)

Сделай 2 экспорта в каждом браузере:

- **Пароли** → экспорт в CSV (потом можно импортировать/залить в менеджер паролей)
- **Закладки/Избранное** → экспорт в HTML

И супер-лайфхак для вкладок:

- Открой все нужные вкладки → **Ctrl+Shift+D** (добавить все вкладки в папку закладок).
Это твой “чекпоинт” сессии без магии.

Уровень С — “полный слепок профиля” (история, куки, сессии, список загрузок)

Это то, что ты просишь “до сессий загрузок”.


ВАЖНО: перед копированием закрой браузеры полностью.

Папки профилей (Chromium-подобные):

- **Edge:**
%LOCALAPPDATA%\Microsoft\Edge\User Data
- **Yandex Browser:**
%LOCALAPPDATA%\Yandex\YandexBrowser\User Data
- **Opera (Stable):**
%APPDATA%\Opera Software\Opera Stable
(иногда ещё полезно сохранить кеш: %LOCALAPPDATA%\Opera Software\Opera Stable)
- **Opera GX:**
%APPDATA%\Opera Software\Opera GX Stable

Что именно внутри важно (если захочешь точно):

Local State , Default/ , Profile * , History (это история+загрузки), Login Data (пароли), Cookies (сессии), Sessions (вкладки), Extensions (расширения).

 **Нюанс про пароли/куки:** они шифруются Windows (DPAPI).

- При **обычном обновлении Win10→Win11 “поверх”** они сохраняются почти всегда.
- При **чистой переустановке** на том же ПК иногда можно восстановить, но часто пароли/куки уже не расшифруются → поэтому экспорт CSV всё равно полезен

как "план Б".

3) Почему "Документы не перенести удалось" и красные крестики

На твоём скрине красные крестики у **OneDrive / Документы / Изображения** — это почти всегда OneDrive "держит" папки (Known Folder Backup) или у него ошибка синхры.

Если хочешь переносить системные папки (Документы/Рабочий стол/Картинки) на D:

1. OneDrive → Settings → **Sync & backup** → **Manage backup**
2. Выключи backup для Documents/Desktop/Pictures
3. Потом уже перенос через: папка → Свойства → **Расположение** → Переместить.

4) Скрипт для бэкапа браузеров (автоматически в D:\Backups)

Сохрани как `browser_backup_win.py` и запусти. Он скопирует профили в одну папку с датой.

python

 Копировать код

```
# browser_backup_win.py
# Backup browser profiles (Edge, Yandex, Opera) to a chosen folder.
# Run AFTER closing browsers for best result.

import os
import shutil
from datetime import datetime
from pathlib import Path

def copytree_fast(src: Path, dst: Path):
    # Python 3.8+ supports dirs_exist_ok
    # We ignore huge caches to keep backup smaller.
    ignore = shutil.ignore_patterns(
        "Cache", "Code Cache", "GPUCache", "ShaderCache",
        "Media Cache", "GrShaderCache", "DawnCache", "Service Worker\\CacheStorag
    )
    shutil.copytree(src, dst, dirs_exist_ok=True, ignore=ignore)

def main():
    local = Path(os.environ.get("LOCALAPPDATA", ""))
```

```

roam = Path(os.environ.get("APPDATA", ""))
user = Path(os.environ.get("USERPROFILE", ""))

stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
# Choose destination: D:\Backups\Browsers\...
dest_root = Path("D:/Backups/Browsers") / f"backup_{stamp}"
dest_root.mkdir(parents=True, exist_ok=True)

sources = {
    "Edge_UserData": local / "Microsoft" / "Edge" / "User Data",
    "Yandex_UserData": local / "Yandex" / "YandexBrowser" / "User Data",
    "Opera_Stable_Roaming": roam / "Opera Software" / "Opera Stable",
    "OperaGX_Stable_Roaming": roam / "Opera Software" / "Opera GX Stable",
    # optional local opera cache/settings (can be big; still useful sometimes
    "Opera_Stable_Local": local / "Opera Software" / "Opera Stable",
    "OperaGX_Stable_Local": local / "Opera Software" / "Opera GX Stable",
}

report = {"saved_to": str(dest_root), "items": []}

print(f"Saving to: {dest_root}")
for name, src in sources.items():
    if src.exists():
        dst = dest_root / name
        print(f"- Copy: {name}\n from: {src}\n to: {dst}")
        try:
            copytree_fast(src, dst)
            report["items"].append({"name": name, "src": str(src), "dst": str(dst)})
        except Exception as e:
            print(f" ! error: {e}")
            report["items"].append({"name": name, "src": str(src), "dst": str(dst), "status": "not"})
    else:
        report["items"].append({"name": name, "src": str(src), "status": "not"})

# save manifest
(dest_root / "manifest.txt").write_text(str(report), encoding="utf-8")
print("\nDone. Manifest saved as manifest.txt")

if __name__ == "__main__":
    main()

```

Запуск:

```
cd $env:USERPROFILE\Desktop  
python .\browser_backup_win.py
```

5) Микро-реальность про Win11 конкретно на твоём ПК

Даже если ты добьёшь место до 30 GB, у тебя всё равно сейчас:

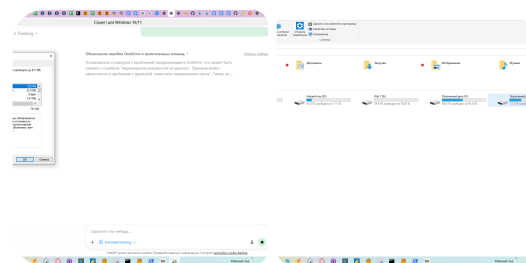
- **Secure Boot = NO**
- **TPM = отсутствует**
- **CPU i3-4170** — официально мимо списка поддержки Win11

То есть “честный апгрейд” Win11 не любит твоё железо. Можно обходить, но это уже режим “я хакер, мама” (и иногда с сюрпризами по обновлениям). Если ты просто хотел освободить место и привести систему в чувство — ты всё сделал правильно.

Если хочешь — скажи, **какой сценарий** ты реально планируешь:

1. оставить Win10 и просто почистить/ускорить
2. ставить Win11 “в обход”
3. переустановка начисто

И я под это дам точный чек-лист бэкапа (включая Telegram/Steam/VS Code/пароли/2FA и т.д.).



и вот уже через другой чат кое что освободил
и вот кстати что из двух чатов в повершеле получил
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все
права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Windows\system32> Get-ChildItem C:\ -Force -
ErrorAction SilentlyContinue |
>> Select-Object Name, @{n="GB";e=
[math]::Round($_.Length/1GB,2)} |
>> Sort-Object GB -Descending | Select-Object -First 20
>>
```

Name	GB
----	--
pagefile.sys	7.16
swapfile.sys	0.02
System Volume Information	0
Users	0
ProgramData	0
Recovery	0
DumpStack.log.tmp	0
\$Recycle.Bin	0
Windows	0
bootTel.dat	0
inetpub	0
MSOCache	0
\$WinREAgent	0
Documents and Settings	0
OneDriveTemp	0
Program Files	0
Program Files (x86)	0
PerfLogs	0
Program Box	0

```
PS C:\Windows\system32> dir C:\ -Force | sort Length -
Descending | select -First 20
>>
```

Каталог: C:\

Mode	LastWriteTime	Length	Name
----	-----	-----	----

```

-a-hs-      12/27/2025  2:47 AM    7685099520
pagefile.sys
-a-hs-      12/27/2025  2:47 AM      16777216
swapfile.sys
-a-hs-      12/27/2025  2:47 AM        8192
DumpStack.log.tmp
---hs-      5/13/2025  10:44 PM      112 bootTel.dat
d--hs-      12/26/2025  11:23 PM      System Volume
Information
d--hs-      1/3/2025   2:48 AM        Recovery
d--h--      12/27/2025  1:51 PM        ProgramData
d--hs-      1/3/2025   2:48 AM        $Recycle.Bin
d-----    10/25/2025  11:34 PM        Windows
d-r---      8/10/2025  11:50 PM        Users
d-----    4/13/2025   7:08 PM        inetpub
d-rh--      3/25/2025   2:25 PM        MSOCache
d--h--      6/11/2025   8:48 PM        $WinREAgent
d--hsl      3/25/2023  10:12 PM        Documents and
Settings
d--h--      1/4/2025   8:32 PM        OneDriveTemp
d-r---      9/9/2025   9:50 PM        Program Files
d-r---      12/27/2025  2:10 PM        Program Files
(x86)
d-----    12/7/2019  12:14 PM        PerfLogs
d-----    3/25/2025   2:23 PM        Program Box

```

```

PS C:\Windows\system32> $root = $env:USERPROFILE
>> Get-ChildItem $root -Directory -Force -ErrorAction
SilentlyContinue |
>>   ForEach-Object {
>>     $sum = (Get-ChildItem $_.FullName -File -Recurse -
Force -ErrorAction SilentlyContinue |
>>       Measure-Object Length -Sum).Sum
>>     [pscustomobject]@{ Path = $_.FullName; GB =
[math]::Round($sum/1GB,2) }
>>   } | Sort-Object GB -Descending | Select-Object -First
20
>>

```

```

Path          GB
-----
C:\Users\Славиус\AppData  17.66

```



```

C:\Users\Славиус\OneDrive    7.69
C:\Users\Славиус\Microsoft   0.81
C:\Users\Славиус\wind surf   0.67
C:\Users\Славиус\jdk s       0.65
C:\Users\Славиус\rustup      0.55
C:\Users\Славиус\codeium     0.52
C:\Users\Славиус\git         0.39
C:\Users\Славиус\IdeaProjects 0.28
C:\Users\Славиус\cache       0.09
C:\Users\Славиус\cursor      0.07
C:\Users\Славиус\Desktop     0.06
C:\Users\Славиус\Fooocus     0.05
C:\Users\Славиус\Recent      0
C:\Users\Славиус\PrintHood    0
C:\Users\Славиус\Music       0
C:\Users\Славиус\NetHood     0
C:\Users\Славиус\codex       0
C:\Users\Славиус\Saved Games  0
C:\Users\Славиус\главное меню 0

```

```

PS C:\Windows\system32> $root = "C:\"
>> Get-ChildItem $root -Directory -Force -ErrorAction
SilentlyContinue |
>>  ForEach-Object {
>>    $sum = (Get-ChildItem $_.FullName -File -Recurse -
Force -ErrorAction SilentlyContinue |
>>      Measure-Object Length -Sum).Sum
>>    [pscustomobject]@{ Path = $_.FullName; GB =
[math]::Round($sum/1GB,2) }
>>  } | Sort-Object GB -Descending | Select-Object -First
20
>>

```

Get-ChildItem : Не удается найти указанный файл
строка:4 знак:13

```

+ $sum = (Get-ChildItem $_.FullName -File -Recurse -
Force -ErrorAct ...
+

```

```

+

```

```

~~~~~
~~~~~

```

```

+ CategoryInfo          : NotSpecified: (:) [Get-ChildItem],
Win32Exception
+ FullyQualifiedErrorId :

```

System.ComponentModel.Win32Exception,Microsoft.PowerShell.Commands.GetChildItemCommand

Path	GB
----	--
C:\Windows	19.81
C:\Program Files	15.34
C:\ProgramData	6.43
C:\Program Files (x86)	2.97
C:\Program Box	0.8
C:\MSOCache	0.74
C:\System Volume Information	0
C:\Recovery	0
C:\Users	0
C:\\$Recycle.Bin	0
C:\\$WinREAgent	0
C:\PerfLogs	0
C:\OneDriveTemp	0
C:\Documents and Settings	0
C:\inetpub	0

```
PS C:\Windows\system32> vssadmin list shadowstorage
>>
```

vssadmin 1.1 - Программа командной строки для администрирования службы теневого копирования томов
(C) Корпорация Майкрософт (Microsoft Corporation), 2001-2013.

Отсутствуют элементы, соответствующие запросу.
PS C:\Windows\system32>

Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Windows\system32> cd $env:USERPROFILE\Desktop
>> python .\win11_readiness_report_v2.py --outdir .
>>
=== Windows 11 Readiness / System Inventory Report
(v2) ===
Generated: 2025-12-27T14:15:56
Host: DESKTOP-1OAT63D | User: Славиус | SystemDrive: C:

--- Windows ---
Product: Windows 10 Home
Version: 2009 | Build: 19045 | Edition: Core
Firmware (from ComputerInfo): 2
Manufacturer/Model: MSI MS-7817

--- CPU / RAM ---
CPU #1: Intel(R) Core(TM) i3-4170 CPU @ 3.70GHz | Cores:
2 | Threads: 4 | AddrWidth: 64
RAM: 11.95 GB

--- Firmware / Security ---
Boot mode: Uefi
Secure Boot: NO
TPM info: {'gettpm': {'TpmPresent': False, 'TpmReady': False,
'TpmEnabled': False, 'TpmActivated': False, 'TpmOwned':
False, 'RestartPending': False, 'ManufacturerId': 0,
'ManufacturerIdTxt': None, 'ManufacturerVersion': None,
'ManufacturerVersionFull20': None, 'ManagedAuthLevel': 4,
'OwnerAuth': None, 'OwnerClearDisabled': True,
'AutoProvisioning': -1, 'LockedOut': False,
'LockoutHealTime': None, 'LockoutCount': None,
'LockoutMax': None, 'SelfTest': None}, 'win32tpm': [{}],
'spec_versions': [], 'tpm2_supported': None, 'error': None}

--- Disks ---
Disk 0: P4-240 | 223.57 GB | Style=GPT | Bus=SATA |
IsSystem=True IsBoot=True
Disk 1: SAMSUNG HD161GJ | 149.05 GB | Style=MBR |
Bus=SATA | IsSystem=False IsBoot=False

--- Volumes ---
G: | FS=NTFS | Label= | Size=24.04 GB | Free=17.57 GB |
Health=Healthy
E: | FS=NTFS | Label=Win 7 | Size=30.01 GB | Free=29.93
```

GB | Health=Healthy
F: | FS=NTFS | Label= | Size=95.0 GB | Free=49.87 GB |
Health=Healthy
C: | FS=NTFS | Label= | Size=111.4 GB | Free=6.78 GB |
Health=Healthy
D: | FS=NTFS | Label=Новый том | Size=111.4 GB |
Free=106.38 GB | Health=Healthy
H: | FS=Unknown | Label= | Size=0.0 GB | Free=0.0 GB |
Health=Healthy

=== Windows 11 Readiness Checks ===

Verdict: NOT READY (some requirements failed)

- [PASS] CPU 64-bit: 64-bit (required: Yes)
- [PASS] RAM: 11.95 GB (required: ≥ 4 GB)
- [PASS] System drive capacity: 111.4 GB (required: ≥ 64 GB (Windows 11 requirement))
- [FAIL] System drive free space: 6.78 GB free (required: ≥ 25 GB (recommended for upgrade headroom))
- [PASS] Boot mode: Uefi (required: UEFI)
- [FAIL] Secure Boot: NO (required: Enabled)
- [FAIL] TPM: Present=NO, Ready=NO, TPM2=UNKNOWN (required: TPM 2.0 present & ready)
- [PASS] System disk partition style: Disk 0: GPT (required: GPT (recommended/typical for UEFI + Win11))

Next steps (based on blockers):

- * Free up space on C: aim for 25–30 GB free before upgrade.
- * Enable Secure Boot in UEFI settings.
- * Enable TPM 2.0 in BIOS/UEFI (Intel PTT / AMD fTPM).

Saved:

-

C:\Users\Славиус\Desktop\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141624.txt

C:\Users\Славиус\Desktop\win11_readiness_v2_DESKTOP-1OAT63D_20251227_141624.json

PS C:\Users\Славиус\Desktop>

PS C:\Users\Славиус\Desktop>

PS C:\Users\Славиус\Desktop>

PS C:\Users\Славиус\Desktop> powercfg -h off

>> DISM /Online /Cleanup-Image

```
/StartComponentCleanup
```

```
>> cleanmgr
```

```
>> clean
```

Deployment Image Servicing and Management tool

Version: 10.0.19041.3636

Image Version: 10.0.19045.6456

```
[===== 10.0% ]  
[=====100.0%=====]  
=====]
```

The operation completed successfully.

clean : Имя "clean" не распознано как имя командлета,
функции, файла сценария или выполняемой
программы. Проверьте

правильность написания имени, а также наличие и
правильность пути, после чего повторите попытку.

строка:4 знак:1

```
+ clean
```

```
+ ~~~~~
```

```
+ CategoryInfo          : ObjectNotFound: (clean:String) [],  
CommandNotFoundException
```

```
+ FullyQualifiedErrorId : CommandNotFoundException
```

```
PS C:\Users\Славиус\Desktop> net stop wuauserv
```

The Центр обновления Windows service is not started.

More help is available by typing NET HELPMSG 3521.

```
PS C:\Users\Славиус\Desktop> net stop bits
```

```
>> ^C
```

```
PS C:\Users\Славиус\Desktop> Remove-Item -Recurse -  
Force "$env:windir\SoftwareDistribution\Download\*" -  
ErrorAction SilentlyContinue
```

```
>> net start bits
```

```
>> net start wuauserv
```

```
>> Clear-RecycleBin -Force -ErrorAction SilentlyContinue
```

```
>> Get-ChildItem $env:TEMP -Force -ErrorAction  
SilentlyContinue | Remove-Item -Recurse -Force -  
ErrorAction SilentlyContinue
```

```
>>
```

The Фоновая интеллектуальная служба передачи (BITS)
service is starting.

The Фоновая интеллектуальная служба передачи (BITS) service was started successfully.

The Центр обновления Windows service is starting.
The Центр обновления Windows service was started successfully.

```
PS C:\Users\Славиус\Desktop> net stop wuauserv
> > net stop bits
> > Remove-Item -Recurse -Force
"$env:windir\SoftwareDistribution\Download\*" -
ErrorAction SilentlyContinue
> > net start bits
> > net start wuauserv
> >
> > Clear-RecycleBin -Force -ErrorAction SilentlyContinue
> > Get-ChildItem $env:TEMP -Force -ErrorAction
SilentlyContinue | Remove-Item -Recurse -Force -
ErrorAction SilentlyContinue
> >
```

The Центр обновления Windows service is stopping.
The Центр обновления Windows service was stopped successfully.

The Фоновая интеллектуальная служба передачи (BITS) service is stopping..

The Фоновая интеллектуальная служба передачи (BITS) service was stopped successfully.

The Фоновая интеллектуальная служба передачи (BITS) service is starting.

The Фоновая интеллектуальная служба передачи (BITS) service was started successfully.

The Центр обновления Windows service is starting.
The Центр обновления Windows service was started successfully.

```
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop>
PS C:\Users\Славиус\Desktop> [math]::Round((Get-
PSDrive C).Free/1GB,2)
```

> >

30.69

PS C:\Users\Славиус\Desktop>

и да

нужно:

оставить Win10 и просто почистить/ускорить

ставить Win11 "в обход"




и потом уже нужен чек лист для бэкапа

Думал на протяжении 1m 38s >

О, вот это уже по-взрослому: **C: свободно 30.69 GB** 

Это ровно то, чего не хватало (и винде стало легче дышать, и апгрейд/обход теперь вообще имеет смысл).

По твоим логам всё ясно:

- UEFI = да, GPT = да 
- Secure Boot = NO 
- TPM = TpmPresent False  (скорее всего TPM реально нет/не поддерживается на этой связке)
- CPU i3-4170 — официально Win11 “не любит”, но “в обход” ставится.

Ты хочешь:

1. оставить Win10 и ускорить
2. Win11 “в обход” (потом)
3. чек-лист бэкапа

Поехали.

1) Win10: чистка + ускорение (без опасных фокусов)

Быстрые победы (минимум риска)

А) Автозагрузка

Ctrl+Shift+Esc → Автозагрузка → вырубь всё, что не нужно на старте (мессенджеры, апдейтеры, лаунчеры).

Сильно влияет на “время до адекватности” после включения.

В) OneDrive (у тебя красные кресты)

Красные кресты = синхра глючит/папки “захвачены” OneDrive.

Если цель — место + стабильность:

- OneDrive → Settings → Sync & backup → Manage backup → отключи backup для Desktop/Documents/Pictures (временно).
- Включи Files On-Demand и сделай большие папки “Online-only” (ПКМ → “Освободить место”).
У тебя OneDrive ~7.7 GB — это отличный резерв, который можно освободить без удаления данных.

С) AppData у тебя 17.66 GB — это главный “жир”

Тут обычно живут кешы:

- Telegram cache
- браузеры (кеш/профили)

- IDE (JetBrains), VS Code, WindSurf, Codeium
- лаунчеры игр

Самое вкусное место для чистки: Telegram и браузеры (и это безопасно).

Telegram Desktop: Settings → Advanced → **Manage local storage** → Clear cache + поставь лимит (например 2–5 GB).

D) SSD-уход (ускоряет “со временем”)

Пуск → “Оптимизация дисков” → выбери C → **Оптимизировать**
(для SSD это TRIM, не дефраг — всё ок)

Что НЕ советую ради “ещё +2 GB”

- Уменьшать/переносить `pagefile.sys` на HDD: да, он 7.16GB, но перенос на медленный диск часто делает систему **тупее** и иногда нестабильнее. Сейчас у тебя 30GB свободно — оставь пейджфайл в покое, он как подушка безопасности.

2) Win11 “в обход” (потом): 2 рабочих пути

Твой железный факт: **TPM нет + CPU не в списке + Secure Boot выключен** → официальный апгрейд будет блокироваться.

Но “в обход” можно.

Вариант 1 (самый простой): USB через Rufus с обходом требований

1. Скачиваешь Win11 ISO
2. Rufus → выбираешь ISO → флешку
3. В Rufus будут галочки типа: **Remove requirement for TPM/Secure Boot/CPU/RAM** → ставишь
4. Грузишься с флешки → ставишь

Плюсы: обычно работает сразу.

Минусы: это чистая установка (если выберешь формат/переразметку — потеря данных; без форматирования можно, но лучше не рисковать).

Вариант 2: In-place upgrade из Windows 10 (с сохранением программ)

Иногда прокатывает мягче.

В PowerShell/ CMD от админа:

```
bat
```

[Копировать код](#)

```
reg add "HKLM\SYSTEM\Setup\MoSetup" /v AllowUpgradesWithUnsupportedTPMOrCPU /t RE
```

Дальше:

- монтируешь ISO Win11 → запускаешь `setup.exe` → “сохранить файлы и приложения”.

Если установщик всё равно упирается, “жёсткий обход” через `LabConfig` обычно делают в среде установщика (`Shift+F10`), но на практике чаще проще Rufus.

Про Secure Boot

Даже если ставишь в обход — **включить Secure Boot полезно** (безопасность + меньше лишних вопросов от инсталлятора).

У тебя уже UEFI+GPT — значит база есть.

В BIOS обычно:

- Disable CSM / Legacy
- Secure Boot → Enable
- OS Type → Windows UEFI Mode

3) Чек-лист бэкапа (перед любыми экспериментами)

3.1. Самое важное “за 15 минут”

Браузеры (Edge/Yandex/Opera)

1. Включи синхронизацию (аккаунты)
2. Экспортируй:
 - Закладки в HTML
 - Пароли в CSV (как план Б)
3. Сохрани “сессию вкладок”:
 - открой нужные вкладки → **Ctrl+Shift+D** (сохранить все вкладки папкой)

Wi-Fi пароли (чтобы не вспоминать)

```
powershell
```

[Копировать код](#)

```
mkdir D:\Backups\WiFi -Force  
netsh wlan export profile key=clear folder="D:\Backups\WiFi"
```

Драйверы (на случай переустановки)

powershell

 Копировать код

```
mkdir D:\Backups\Drivers -Force  
dism /online /export-driver /destination:D:\Backups\Drivers
```

3.2. Папки, которые реально спасают жизнь

Скопировать на D:\Backups (или на внешний диск):

Профиль пользователя

- C:\Users\Славиус\Desktop
- C:\Users\Славиус\Documents
- C:\Users\Славиус\Pictures
- C:\Users\Славиус\Downloads (если там ещё что-то осталось)
- C:\Users\Славиус\OneDrive (или убедиться, что всё синхронизировано)

Код/проекты

- C:\Users\Славиус\My-Projects (и любые IdeaProjects/репы)
- C:\Users\Славиус\.ssh (ключи!)
- .gitconfig (если есть)
- папки с рабочими скриптами/архивами

Telegram Desktop (если важны сессии/секретные чаты)

- %APPDATA%\Telegram Desktop\tdata
(обычные чаты в облаке, но сессия/секретные — локально)

VS Code

- %APPDATA%\Code\User\ (settings.json, snippets и т.д.)
- список расширений:

powershell

 Копировать код

```
code --list-extensions > D:\Backups\vscode_extensions.txt
```

JetBrains (если используешь IntelliJ/PyCharm)

Обычно:

- %APPDATA%\JetBrains\
 - %LOCALAPPDATA%\JetBrains\
- (в зависимости от версии)
-

3.3. “Полный слепок браузеров” (история/куки/сессии)

Ты уже просил — повторю коротко и по делу:

копируешь целиком профили (после закрытия браузеров):

- Edge: %LOCALAPPDATA%\Microsoft\Edge\User Data
- Yandex: %LOCALAPPDATA%\Yandex\YandexBrowser\User Data
- Opera: %APPDATA%\Opera Software\Opera Stable (+ иногда %LOCALAPPDATA%\Opera Software\Opera Stable)

(Я тебе уже дал скрипт бэкапа браузеров — он как раз это делает.)


4) По твоим данным: где ещё можно выжать место без боли

Из твоего вывода:

- AppData 17.66 GB — главная цель (кеши)
- OneDrive 7.69 GB — можно сделать online-only
- pagefile.sys 7.16 GB — **не трогать** ради скорости/стабильности

Если хочешь, я дам **точечный PowerShell**, который покажет ТОП-папок внутри AppData (Local/Roaming) — и мы выкинем мусор так, чтобы винда не обиделась и не ушла в депрессию.

Что делать прямо сейчас (короткий план)

1. Раз ты уже на **30GB free** — красавец, цель по месту выполнена 
2. Почистить **Telegram cache** + **браузерные кеши** (самый безопасный профит)
3. OneDrive: включить **Files On-Demand** + “Online-only” на тяжёлых папках
4. Сделать **бэкап по чек-листу** в D:\Backups\...
5. Уже потом — думать про Win11 “в обход”.

Хочешь — напиши, каким браузером пользуешься как основным (Яндекс/Опера/Эдж), и пользуешься ли Telegram Desktop активно. Я под это дам **самый короткий и безопасный "пакет чистки"**, чтобы не тронуть ничего ценного.



Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Windows\system32> # Т папок в Local (кто жрёт 14.5 )
PS C:\Windows\system32> $root = "$env:USERPROFILE\AppData\Local"
PS C:\Windows\system32> Get-ChildItem $root -Directory -Force -ErrorAction SilentlyContinue |
>>   ForEach-Object {
>>     $sum = (Get-ChildItem $_.FullName -File -Recurse -Force -ErrorAction SilentlyContinue |
>>       Measure-Object Length -Sum).Sum
>>     [pscustomobject]@{ Path = $_.FullName; GB = [math]::Round($sum/1GB,2) }
>>   } | Sort-Object GB -Descending | Select-Object -First 25
```

Path	GB
----	--
C:\Users\Славиус\AppData\Local\Yandex	3.11
C:\Users\Славиус\AppData\Local\Programs	2.48
C:\Users\Славиус\AppData\Local\Microsoft	2.43
C:\Users\Славиус\AppData\Local\Packages	1.92
C:\Users\Славиус\AppData\Local\JetBrains	1.2
C:\Users\Славиус\AppData\Local\GitHubDesktop	1.01
C:\Users\Славиус\AppData\Local\SquirrelTemp	0.65
C:\Users\Славиус\AppData\Local\Opera Software	0.45

C:\Users\Славиус\AppData\Local\Steam	0.29
C:\Users\Славиус\AppData\Local\puccinialin	0.19
C:\Users\Славиус\AppData\Local\pip	0.18
C:\Users\Славиус\AppData\Local\pachca-updater	0.15
C:\Users\Славиус\AppData\Local\Temp	0.14
C:\Users\Славиус\AppData\Local\ms-playwright-go	0.09
C:\Users\Славиус\AppData\Local\CrashDumps	0.07
C:\Users\Славиус\AppData\Local\NVIDIA Corporation	0.04
C:\Users\Славиус\AppData\Local\Package Cache	0.03
C:\Users\Славиус\AppData\Local\Comms	0.03
C:\Users\Славиус\AppData\Local\CEF	0.02
C:\Users\Славиус\AppData\Local\рура	0.01
C:\Users\Славиус\AppData\Local\PlaceholderTileLogoFolder	0
C:\Users\Славиус\AppData\Local\Rockstar Games	0
C:\Users\Славиус\AppData\Local\PAYDAY 2	0
C:\Users\Славиус\AppData\Local\seaborn	0
C:\Users\Славиус\AppData\Local\TunnelBear	0

PS C:\Windows\system32>

PS C:\Windows\system32> # ТОП папок в Roaming (кто жрёт 3 ГБ)

```
>> $root = "$env:USERPROFILE\AppData\Roaming"
>> Get-ChildItem $root -Directory -Force -ErrorAction SilentlyContinue |
>>   ForEach-Object {
>>     $sum = (Get-ChildItem $_.FullName -File -Recurse -Force -ErrorAction SilentlyContinue |
>>       Measure-Object Length -Sum).Sum
>>     [pscustomobject]@{ Path = $_.FullName; GB = [math]::Round($sum/1GB,2) }
>>   } | Sort-Object GB -Descending | Select-Object -First 25
>>
```

Path

GB

```

-----
C:\Users\Славиус\AppData\Roaming\JetBrains      1.24
C:\Users\Славиус\AppData\Roaming\Windsurf      0.58
C:\Users\Славиус\AppData\Roaming\Opera Software 0.56
C:\Users\Славиус\AppData\Roaming\TunnelBear    0.18
C:\Users\Славиус\AppData\Roaming\Microsoft    0.14
C:\Users\Славиус\AppData\Roaming\Python        0.09
C:\Users\Славиус\AppData\Roaming\Cursor        0.08
C:\Users\Славиус\AppData\Roaming\Pachca        0.03
C:\Users\Славиус\AppData\Roaming\utorrent      0.03
C:\Users\Славиус\AppData\Roaming\WinRAR        0.02
C:\Users\Славиус\AppData\Roaming\GitHub Desktop
0.01
C:\Users\Славиус\AppData\Roaming\DBeaverData   0.01
C:\Users\Славиус\AppData\Roaming\Yandex        0
C:\Users\Славиус\AppData\Roaming\VK            0
C:\Users\Славиус\AppData\Roaming\VK Messenger  0
C:\Users\Славиус\AppData\Roaming\jupyter       0
C:\Users\Славиус\AppData\Roaming\Electronic Arts 0
C:\Users\Славиус\AppData\Roaming\AnyDesk       0
C:\Users\Славиус\AppData\Roaming\postgresql   0
C:\Users\Славиус\AppData\Roaming\PolarBear     0
C:\Users\Славиус\AppData\Roaming\Adobe        0

```

PS C:\Windows\system32>

```

PS C:\Windows\system32> Remove-Item -Recurse -Force
"C:\ProgramData\NVIDIA Corporation\Downloader\*" -
ErrorAction SilentlyContinue
>> Remove-Item -Recurse -Force
"C:\ProgramData\NVIDIA Corporation\NV_Cache\*" -
ErrorAction SilentlyContinue
>>
PS C:\Windows\system32>

```

и да
потом можешь написать объединенный скрипт на
обнаружение
очистку мб
или анализ

а так же настройку и тд
или скинешь все архивом
с пометко что есть что

Думал на протяжении 1m 3s >

Анализ ошибочен ^

python

```
import os, textwrap, zipfile, da
from datetime import datetime
base_dir = "/mnt/data/win_maintain_pack"
os.makedirs(base_dir, exist_ok=True)

win_maintain_py = r'''# win_maintain.py
# Windows 10/11 maintenance helper: scan disk hogs, backup browser profiles, and
# No external deps. Python 3.9+ recommended.
#
# Safety:
# - Default cleanup mode is DRY-RUN (shows what would be deleted).
# - Only deletes known cache/temp locations unless you explicitly enable extra fl
# - Skips symlinks/reparse points while scanning to avoid loops.
#
# Usage examples:
# python win_maintain.py scan
# python win_maintain.py scan --roots "%LOCALAPPDATA%" "%APPDATA%"
# python win_maintain.py cleanup (dry-run)
# python win_maintain.py cleanup --yes (execute)
```

Всегда показывать подробности

Копировать код


```
# python win_maintain.py cleanup --yes --browser-cache (also clear browser ca
# python win_maintain.py backup-browsers --dest D:\Backups\Browsers
# python win_maintain.py winupdate-cache --yes
#
# Tip: run PowerShell as Administrator for best results.
```

```
from __future__ import annotations
```

```
import argparse
import csv
import ctypes
import heapq
import json
import os
import shutil
import subprocess
import sys
import time
from dataclasses import dataclass, asdict
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Optional, Tuple
```

```
REPARSE_POINT_ATTR = 0x0400 # FILE_ATTRIBUTE_REPARSE_POINT
```

```
def is_admin() -> bool:
    try:
        return bool(ctypes.windll.shell32.IsUserAnAdmin())
    except Exception:
        return False
```

```
def decode_best_effort(b: bytes) -> str:
    if b is None:
        return ""
    for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
        try:
            return b.decode(enc)
        except Exception:
            continue
    return b.decode("utf-8", errors="replace")
```

```
def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
    p = subprocess.run(cmd, capture_output=True)
    out = decode_best_effort(p.stdout).strip()
    err = decode_best_effort(p.stderr).strip()
    return p.returncode, out, err

def run_powershell(ps: str) -> Tuple[int, str, str]:
    prefix = "[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new(); $Pro"
    return run_cmd(["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-C"

def format_gb(num_bytes: float) -> str:
    return f"{num_bytes / (1024**3):.2f} GB"

def is_reparse_point(path: str) -> bool:
    try:
        st = os.stat(path, follow_symlinks=False)
        attrs = getattr(st, "st_file_attributes", 0)
        return bool(attrs & REPARSE_POINT_ATTR)
    except Exception:
        return False

def list_drives() -> List[str]:
    drives = []
    for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        p = f"{c}:\\"
        if os.path.exists(p):
            drives.append(p)
    return drives

@dataclass
class ScanStats:
    root: str
    depth: int
    files_scanned: int = 0
    dirs_scanned: int = 0
    bytes_total: int = 0
```

```
errors: int = 0
denied: int = 0
skipped_reparse: int = 0
elapsed_sec: float = 0.0
```

```
@dataclass
```

```
class ScanResult:
```

```
    stats: ScanStats
    top_dirs_level1: List[Tuple[str, int]]
    top_dirs_level2: List[Tuple[str, int]]
    top_files: List[Tuple[str, int]]
```

```
def scan_root(root: str, depth: int, top_dirs: int, top_files: int) -> ScanResult
```

```
    t0 = time.time()
    root = os.path.abspath(root)
    stats = ScanStats(root=root, depth=depth)
```

```
    lvl1: Dict[str, int] = {}
    lvl2: Dict[str, int] = {}
    file_heap: List[Tuple[int, str]] = []
```

```
    stack = [root]
    root_parts_len = len(Path(root).parts)
```

```
    def add_file(sz: int, p: str):
        if top_files <= 0:
            return
        if len(file_heap) < top_files:
            heapq.heappush(file_heap, (sz, p))
        else:
            if sz > file_heap[0][0]:
                heapq.heapreplace(file_heap, (sz, p))
```

```
    while stack:
        cur = stack.pop()
        stats.dirs_scanned += 1
        try:
            with os.scandir(cur) as it:
                for entry in it:
                    try:
                        if entry.is_symlink():
```

```
        continue
    full = entry.path
    if is_reparse_point(full):
        stats.skipped_reparse += 1
        continue

    if entry.is_dir(follow_symlinks=False):
        rel_parts = len(Path(full).parts) - root_parts_len
        if depth < 0 or rel_parts <= depth:
            stack.append(full)
    elif entry.is_file(follow_symlinks=False):
        st = entry.stat(follow_symlinks=False)
        sz = int(getattr(st, "st_size", 0))
        stats.files_scanned += 1
        stats.bytes_total += sz

        try:
            rel = os.path.relpath(full, root)
        except Exception:
            rel = full
        parts = rel.split(os.sep)
        if len(parts) >= 1:
            lvl1[parts[0]] = lvl1.get(parts[0], 0) + sz
        if len(parts) >= 2:
            k2 = os.path.join(parts[0], parts[1])
            lvl2[k2] = lvl2.get(k2, 0) + sz

        add_file(sz, full)
    except PermissionError:
        stats.denied += 1
    except FileNotFoundError:
        stats.errors += 1
    except OSError:
        stats.errors += 1
except PermissionError:
    stats.denied += 1
except FileNotFoundError:
    stats.errors += 1
except OSError:
    stats.errors += 1

stats.elapsed_sec = round(time.time() - t0, 2)
top1 = sorted(lvl1.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
```

```

top2 = sorted(lvl2.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top_files_list = sorted([(p, sz) for (sz, p) in file_heap], key=lambda x: x[1]
return ScanResult(stats=stats, top_dirs_level1=top1, top_dirs_level2=top2, to

```

```

def print_drive_table() -> List[Dict[str, str]]:
    rows = []
    print("=== Drives ===")
    for d in list_drives():
        try:
            du = shutil.disk_usage(d)
            used = du.total - du.free
            print(f"{d} Total: {format_gb(du.total)} | Used: {format_gb(used)} |")
            rows.append({"drive": d, "total_gb": f"{du.total/(1024**3):.2f}", "fr
        except Exception as e:
            print(f"{d} (error reading usage: {e})")
            rows.append({"drive": d, "error": str(e)})
    print()
    return rows

```

```
@dataclass
```

```
class CleanupAction:
```

```

    name: str
    description: str
    # list of paths/globs to delete
    targets: List[str]
    # whether action needs admin
    needs_admin: bool = False

```

```
def expand_globs(patterns: List[str]) -> List[Path]:
```

```

    out: List[Path] = []
    for pat in patterns:
        # environment variables in string
        pat2 = os.path.expandvars(pat)
        # support trailing \* and basic wildcards via glob
        if "*" in pat2 or "?" in pat2:
            # Path.glob doesn't like absolute with drive wildcards; use glob from
            p = Path(pat2)
            # If wildcard in name, split
            if any(ch in p.name for ch in "*?"):
                parent = p.parent

```

```

        if parent.exists():
            out.extend(list(parent.glob(p.name)))
        else:
            out.append(p)
    else:
        out.append(Path(pat2))
# de-dup
uniq = []
seen = set()
for p in out:
    s = str(p).lower()
    if s not in seen:
        uniq.append(p)
        seen.add(s)
return uniq

def remove_path(p: Path) -> Tuple[bool, str]:
    try:
        if not p.exists():
            return True, "not_found"
        # don't follow symlinks/reparse points when deleting directories
        if p.is_symlink():
            return True, "skip_symlink"
        if is_reparse_point(str(p)):
            return True, "skip_reparse_point"
        if p.is_file():
            p.unlink()
            return True, "deleted_file"
        if p.is_dir():
            shutil.rmtree(p, ignore_errors=False)
            return True, "deleted_dir"
        return True, "unknown_type"
    except PermissionError as e:
        return False, f"permission_denied: {e}"
    except Exception as e:
        return False, f"error: {e}"

def cleanup(actions: List[CleanupAction], yes: bool, outdir: Path) -> Dict[str, o
report = {
    "generated_at": datetime.now().isoformat(timespec="seconds"),
    "is_admin": is_admin(),

```

```

        "mode": "EXECUTE" if yes else "DRY_RUN",
        "actions": [],
    }

    for act in actions:
        act_entry = {"name": act.name, "description": act.description, "needs_admin":
            if act.needs_admin and not is_admin():
                act_entry["skipped"] = "needs_admin"
                report["actions"].append(act_entry)
                continue

        paths = expand_globs(act.targets)
        for p in paths:
            item = {"path": str(p), "exists": p.exists()}
            if yes:
                ok, msg = remove_path(p)
                item["result"] = msg
                item["ok"] = ok
            act_entry["items"].append(item)

        report["actions"].append(act_entry)

    # Also clear recycle bin via PowerShell if executing
    if yes:
        code, out, err = run_powershell("try { Clear-RecycleBin -Force -ErrorAction SilentlyContinue }")
        report["recycle_bin"] = {"code": code, "out": out, "err": err}

    # Save
    outdir.mkdir(parents=True, exist_ok=True)
    ts = datetime.now().strftime("%Y%m%d_%H%M%S")
    path = outdir / f"cleanup_report_{ts}.json"
    path.write_text(json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8")
    return report

def winupdate_cache_reset(yes: bool, outdir: Path) -> Dict[str, object]:
    """
    Safe reset of Windows Update download cache.
    Deletes only SoftwareDistribution\Download contents (already standard).
    """
    report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "mode": "EXECUTE",
        "target": os.path.expandvars(r"%windir%\SoftwareDistribution\Download\*")}
    report["target"] = target

```

```

if not yes:
    report["note"] = "Would stop wuauserv/bits, delete Download cache, then s
    (outdir / f"winupdate_cache_dryrun_{datetime.now().strftime('%Y%m%d_%H%M%S')}
        json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
    )
    return report

if not is_admin():
    report["error"] = "Run as Administrator for this action."
    (outdir / f"winupdate_cache_error_{datetime.now().strftime('%Y%m%d_%H%M%S')}
        json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
    )
    return report

cmds = [
    ["net", "stop", "wuauserv"],
    ["net", "stop", "bits"],
    ["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-Command",
     r"Remove-Item -Recurse -Force "$env:windir\SoftwareDistribution\Downloa
    ["net", "start", "bits"],
    ["net", "start", "wuauserv"],
]
results = []
for c in cmds:
    code, out, err = run_cmd(c)
    results.append({"cmd": " ".join(c), "code": code, "out": out, "err": err})
report["results"] = results

(outdir / f"winupdate_cache_{datetime.now().strftime('%Y%m%d_%H%M%S')}").json")
    json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
return report

```

```
def backup_browsers(dest: str) -> Dict[str, object]:
```

```
    """
```

```
    Backup browser profiles (Edge/Yandex/Opera) to destination folder.
```

```
    We ignore heavy caches for smaller backups.
```

```
    """
```

```
    local = Path(os.environ.get("LOCALAPPDATA", ""))
```

```
    roam = Path(os.environ.get("APPDATA", ""))
```

```
    dest_root = Path(dest).expanduser().resolve()
```



```

stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
out = dest_root / f"browsers_backup_{stamp}"
out.mkdir(parents=True, exist_ok=True)

def copytree_fast(src: Path, dst: Path):
    ignore = shutil.ignore_patterns(
        "Cache", "Code Cache", "GPUCache", "ShaderCache",
        "Media Cache", "GrShaderCache", "DawnCache",
        "Service Worker\\CacheStorage", "Service Worker\\ScriptCache",
        "Crashpad", "Crash Reports"
    )
    shutil.copytree(src, dst, dirs_exist_ok=True, ignore=ignore)

sources = {
    "Edge_UserData": local / "Microsoft" / "Edge" / "User Data",
    "Yandex_UserData": local / "Yandex" / "YandexBrowser" / "User Data",
    "Opera_Stable_Roaming": roam / "Opera Software" / "Opera Stable",
    "OperaGX_Stable_Roaming": roam / "Opera Software" / "Opera GX Stable",
}

report = {"saved_to": str(out), "items": []}
for name, src in sources.items():
    if src.exists():
        dst = out / name
        try:
            copytree_fast(src, dst)
            report["items"].append({"name": name, "src": str(src), "dst": str(dst)})
        except Exception as e:
            report["items"].append({"name": name, "src": str(src), "dst": str(dst), "error": str(e)})
    else:
        report["items"].append({"name": name, "src": str(src), "status": "not found"})

(out / "manifest.json").write_text(json.dumps(report, ensure_ascii=False, indent=4))
return report

def optimize_ssd(drive_letter: str) -> Dict[str, object]:
    """
    Trigger TRIM (Optimize-Volume) for SSD. Needs admin.
    """
    report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "drive": drive_letter}
    if not is_admin():
        report["error"] = "Run as Administrator for TRIM."

```

```

        return report

    ps = f"try {{ Optimize-Volume -DriveLetter {drive_letter} -ReTrim -Verbose }}"
    code, out, err = run_powershell(ps)
    report.update({"code": code, "out": out, "err": err})
    return report


def cmd_scan(args: argparse.Namespace):
    drives_info = print_drive_table()

    roots = args.roots or []
    if not roots:
        user = os.environ.get("USERPROFILE", "")
        roots = [
            os.path.join(user, "AppData", "Local"),
            os.path.join(user, "AppData", "Roaming"),
            os.path.join(os.environ.get("windir", r"C:\Windows"), "SoftwareDistri
            os.path.join(os.environ.get("windir", r"C:\Windows"), "Temp"),
        ]

    results: List[ScanResult] = []
    for r in roots:
        if not r:
            continue
        r = os.path.expandvars(r)
        if not os.path.exists(r):
            print(f"[skip] Not found: {r}")
            continue
        print(f"=== Scanning: {r} (depth={args.depth}) ===")
        res = scan_root(r, depth=args.depth, top_dirs=args.top, top_files=args.fi
        results.append(res)

        st = res.stats
        print(f"Scanned dirs={st.dirs_scanned}, files={st.files_scanned}, total={
            f"denied={st.denied}, errors={st.errors}, skipped_reparse={st.skipp

    print("--- Top folders (level 1) ---")
    for name, sz in res.top_dirs_level1:
        print(f"{name:<45} {format_gb(sz)}")
    print("\n--- Top folders (level 2) ---")
    for name, sz in res.top_dirs_level2:
        print(f"{name:<45} {format_gb(sz)}")
    print("\n--- Top files ---")

```

```

    for p, sz in res.top_files:
        print(f"{format_gb(sz):>10}  {p}")
    print("\n" + "=" * 70 + "\n")

outdir = Path(args.outdir).resolve()
outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base = outdir / f"scan_report_{ts}"

# JSON
payload = {
    "generated_at": datetime.now().isoformat(timespec="seconds"),
    "is_admin": is_admin(),
    "drives": drives_info,
    "results": [
        {"stats": asdict(r.stats), "top_dirs_level1": r.top_dirs_level1, "top.
        for r in results
    ],
}
(base.with_suffix(".json")).write_text(json.dumps(payload, ensure_ascii=False)

# CSV for dirs
with open(base.with_suffix("_topdirs.csv"), "w", encoding="utf-8-sig", newlin
    w = csv.writer(f, delimiter=";")
    w.writerow(["root", "level", "path", "bytes", "gb"])
    for r in results:
        root = r.stats.root
        for p, sz in r.top_dirs_level1:
            w.writerow([root, 1, p, sz, round(sz / (1024**3), 3)])
        for p, sz in r.top_dirs_level2:
            w.writerow([root, 2, p, sz, round(sz / (1024**3), 3)])

print(f"Saved:\n- {base.with_suffix('.json')}\n- {base.with_suffix('_topdirs.

def build_cleanup_actions(include_browser_cache: bool) -> List[CleanupAction]:
    local = r"%LOCALAPPDATA%"
    roam = r"%APPDATA%"
    temp = r"%TEMP%"
    windir = r"%windir%"

    actions = [
        CleanupAction(

```

```

        name="Temp (user)",
        description="Clear user TEMP files (safe).",
        targets=[temp + r"\*"],
        needs_admin=False,
    ),
    CleanupAction(
        name="CrashDumps",
        description="Clear Windows Error Reporting crash dumps for user (safe)",
        targets=[local + r"\CrashDumps\*"],
        needs_admin=False,
    ),
    CleanupAction(
        name="SquirrelTemp",
        description="Clear installer temp used by Squirrel-based updaters (safe)",
        targets=[local + r"\SquirrelTemp\*"],
        needs_admin=False,
    ),
    CleanupAction(
        name="NVIDIA cache (ProgramData)",
        description="Clear NVIDIA downloader + NV_Cache (safe).",
        targets=[r"C:\ProgramData\NVIDIA Corporation\Downloader\*", r"C:\ProgramData\NVIDIA Corporation\NV_Cache\*"],
        needs_admin=True,
    ),
]

```

```

if include_browser_cache:
    # Only cache folders; DOES NOT delete profiles (history/passwords/cookies)
    actions.extend([
        CleanupAction(
            name="Yandex Browser cache",
            description="Clear Yandex browser caches only (close browser first)",
            targets=[
                local + r"\Yandex\YandexBrowser\User Data\*\Cache\*",
                local + r"\Yandex\YandexBrowser\User Data\*\Code Cache\*",
                local + r"\Yandex\YandexBrowser\User Data\*\GPUCache\*",
                local + r"\Yandex\YandexBrowser\User Data\*\Service Worker\Cache\*",
                local + r"\Yandex\YandexBrowser\User Data\*\Service Worker\Scripts\*",
            ],
            needs_admin=False,
        ),
        CleanupAction(
            name="Opera cache",
            description="Clear Opera caches only (close browser first).",

```

```

        targets=[
            local + r"\Opera Software\Opera Stable\Cache\*",
            local + r"\Opera Software\Opera Stable\Code Cache\*",
            local + r"\Opera Software\Opera Stable\GPUCache\*",
            local + r"\Opera Software\Opera GX Stable\Cache\*",
            local + r"\Opera Software\Opera GX Stable\Code Cache\*",
            local + r"\Opera Software\Opera GX Stable\GPUCache\*",
        ],
        needs_admin=False,
    ),
    CleanupAction(
        name="Edge cache",
        description="Clear Edge caches only (close browser first).",
        targets=[
            local + r"\Microsoft\Edge\User Data\*\Cache\*",
            local + r"\Microsoft\Edge\User Data\*\Code Cache\*",
            local + r"\Microsoft\Edge\User Data\*\GPUCache\*",
            local + r"\Microsoft\Edge\User Data\*\Service Worker\CacheSto
            local + r"\Microsoft\Edge\User Data\*\Service Worker\ScriptCa
        ],
        needs_admin=False,
    ),
])

return actions

def cmd_cleanup(args: argparse.Namespace):
    outdir = Path(args.outdir).resolve()
    actions = build_cleanup_actions(include_browser_cache=args.browser_cache)

    print(f"Mode: {'EXECUTE' if args.yes else 'DRY_RUN'}")
    if args.browser_cache:
        print("Including browser caches: YES (close browsers before executing)")

    rep = cleanup(actions, yes=args.yes, outdir=outdir)
    # brief summary to console
    print(f"Report saved into: {outdir}")
    if rep.get("mode") == "DRY_RUN":
        print("Dry-run complete. Re-run with --yes to actually delete.")
    else:
        print("Cleanup executed.")

```

```
def cmd_backup(args: argparse.Namespace):
    rep = backup_browsers(args.dest)
    print("Backup complete:")
    print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_winupdate_cache(args: argparse.Namespace):
    outdir = Path(args.outdir).resolve()
    rep = winupdate_cache_reset(yes=args.yes, outdir=outdir)
    print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_trim(args: argparse.Namespace):
    rep = optimize_ssd(args.drive)
    print(json.dumps(rep, ensure_ascii=False, indent=2))

def main():
    if os.name != "nt":
        print("This tool is for Windows.")
        sys.exit(1)

    ap = argparse.ArgumentParser("win_maintain", description="Scan + safe cleanup")
    ap.add_argument("--outdir", default=".", help="Output dir for reports (default: .)")

    sub = ap.add_subparsers(dest="cmd", required=True)

    sp_scan = sub.add_parser("scan", help="Scan disk hogs under common roots (or")
    sp_scan.add_argument("--roots", nargs="*", default=[], help="Roots to scan (default: .)")
    sp_scan.add_argument("--depth", type=int, default=6, help="Relative depth (default: 6)")
    sp_scan.add_argument("--top", type=int, default=25, help="Top N folders (default: 25)")
    sp_scan.add_argument("--files", type=int, default=30, help="Top N files (default: 30)")
    sp_scan.set_defaults(func=cmd_scan)

    sp_cleanup = sub.add_parser("cleanup", help="Safe cleanup (temp/caches). Default: .")
    sp_cleanup.add_argument("--yes", action="store_true", help="Actually delete (default: False)")
    sp_cleanup.add_argument("--browser-cache", action="store_true", help="Also cleanup browser cache (default: False)")
    sp_cleanup.set_defaults(func=cmd_cleanup)

    sp_backup = sub.add_parser("backup-browsers", help="Backup browser profiles to")
    sp_backup.add_argument("--dest", default=r"D:\Backups\Browsers", help="Destination directory (default: D:\Backups\Browsers)")
    sp_backup.set_defaults(func=cmd_backup)
```

```

sp_wu = sub.add_parser("winupdate-cache", help="Reset Windows Update download
sp_wu.add_argument("--yes", action="store_true", help="Execute (otherwise dry
sp_wu.set_defaults(func=cmd_winupdate_cache)

sp_trim = sub.add_parser("trim", help="Run SSD TRIM (Optimize-Volume) for a d
sp_trim.add_argument("--drive", default="C", help="Drive letter (default C).")
sp_trim.set_defaults(func=cmd_trim)

args = ap.parse_args()
args.func(args)

if __name__ == "__main__":
    main()
...

readme = r'''# win_maintain_pack

```

Набор скриптов для Windows 10/11: анализ «что жрёт место», безопасная очистка кэш

Содержимое

- `win_maintain.py` – основной «комбайн»:
- `scan` – сканирует выбранные корни и показывает ТОП папок/файлов
- `cleanup` – безопасная очистка (по умолчанию dry-run)
- `cleanup --browser-cache` – чистит только кэши браузеров (профили/пароли/исто
- `backup-browsers` – бэкап профилей Edge/Yandex/Opera (без тяжёлых кэшей)
- `winupdate-cache` – очистка кэша скачанных обновлений Windows Update
- `trim` – TRIM для SSD (Optimize-Volume)

Запуск

Открой PowerShell (желательно **от имени администратора**), перейди в папку со ск

```

```powershell
python .\win_maintain.py scan --outdir .
python .\win_maintain.py cleanup --outdir . # dry-run
python .\win_maintain.py cleanup --yes --outdir . # выполнить
python .\win_maintain.py cleanup --yes --browser-cache --outdir .
python .\win_maintain.py backup-browsers --dest D:\Backups\Browsers
python .\win_maintain.py winupdate-cache --yes --outdir .
python .\win_maintain.py trim --drive C

```

## Важно

- Для `cleanup --browser-cache` **закрой** браузеры перед запуском (иначе часть файлов будет занята).
- Скрипт **не удаляет** профили браузеров и не трогает пароли/куки/историю (чистит только `Cache/Code Cache/GPUCache/Service Worker CacheStorage`).
- Папки `AppData\Local\Programs` , `AppData\Local\Packages` , `AppData\Local\Microsoft` — это не «мусор». Их надо чистить точечно. Скрипт их не трогает.
- ...

## Write files

```
with open(os.path.join(base_dir, "win_maintain.py"), "w", encoding="utf-8") as f:
 f.write(win_maintain_py)

with open(os.path.join(base_dir, "README.md"), "w", encoding="utf-8") as f:
 f.write(readme)

zip_path = "/mnt/data/win_maintain_pack.zip"
with zipfile.ZipFile(zip_path, "w", compression=zipfile.ZIP_DEFLATED) as z:
 for fn in ["win_maintain.py", "README.md"]:
 z.write(os.path.join(base_dir, fn), arcname=fn)

zip_path
```

[Всегда показывать подробности](#)[Копировать код](#)

Проанализировано ^

python

```
import os, zipfile
from datetime import datetime
```

[Всегда показывать подробности](#)[Копировать код](#)

```
base_dir = "/mnt/data/win_maintain_pack"
os.makedirs(base_dir, exist_ok=True)
```

```
win_maintain_py = r'''# win_maintain.py
Windows 10/11 maintenance helper: scan disk hogs, backup browser profiles, and
No external deps. Python 3.9+ recommended.
#
```



```
Safety:
- Default cleanup mode is DRY-RUN (shows what would be deleted).
- Only deletes known cache/temp locations unless you explicitly enable extra fl
- Skips symlinks/reparse points while scanning to avoid loops.
#
Usage examples:
python win_maintain.py scan
python win_maintain.py scan --roots "%LOCALAPPDATA%" "%APPDATA%"
python win_maintain.py cleanup (dry-run)
python win_maintain.py cleanup --yes (execute)
python win_maintain.py cleanup --yes --browser-cache (also clear browser ca
python win_maintain.py backup-browsers --dest D:\Backups\Browsers
python win_maintain.py winupdate-cache --yes
#
Tip: run PowerShell as Administrator for best results.
```

```
from __future__ import annotations
```

```
import argparse
import csv
import ctypes
import heapq
import json
import os
import shutil
import subprocess
import sys
import time
from dataclasses import dataclass, asdict
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Tuple
```

```
REPARSE_POINT_ATTR = 0x0400 # FILE_ATTRIBUTE_REPARSE_POINT
```

```
def is_admin() -> bool:
 try:
 return bool(ctypes.windll.shell32.IsUserAnAdmin())
 except Exception:
 return False
```

```
def decode_best_effort(b: bytes) -> str:
 if b is None:
 return ""
 for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
 try:
 return b.decode(enc)
 except Exception:
 continue
 return b.decode("utf-8", errors="replace")

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
 p = subprocess.run(cmd, capture_output=True)
 out = decode_best_effort(p.stdout).strip()
 err = decode_best_effort(p.stderr).strip()
 return p.returncode, out, err

def run_powershell(ps: str) -> Tuple[int, str, str]:
 prefix = "[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new(); $Pro"
 return run_cmd(["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-C"

def format_gb(num_bytes: float) -> str:
 return f"{num_bytes / (1024**3):.2f} GB"

def is_reparse_point(path: str) -> bool:
 try:
 st = os.stat(path, follow_symlinks=False)
 attrs = getattr(st, "st_file_attributes", 0)
 return bool(attrs & REPARSE_POINT_ATTR)
 except Exception:
 return False

def list_drives() -> List[str]:
 drives = []
 for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
 p = f"{c}:\\"
 if os.path.exists(p):
 drives.append(p)
```

```
return drives
```

```
@dataclass
```

```
class ScanStats:
```

```
 root: str
```

```
 depth: int
```

```
 files_scanned: int = 0
```

```
 dirs_scanned: int = 0
```

```
 bytes_total: int = 0
```

```
 errors: int = 0
```

```
 denied: int = 0
```

```
 skipped_reparse: int = 0
```

```
 elapsed_sec: float = 0.0
```

```
@dataclass
```

```
class ScanResult:
```

```
 stats: ScanStats
```

```
 top_dirs_level1: List[Tuple[str, int]]
```

```
 top_dirs_level2: List[Tuple[str, int]]
```

```
 top_files: List[Tuple[str, int]]
```

```
def scan_root(root: str, depth: int, top_dirs: int, top_files: int) -> ScanResult
```

```
 t0 = time.time()
```

```
 root = os.path.abspath(root)
```

```
 stats = ScanStats(root=root, depth=depth)
```

```
 lvl1: Dict[str, int] = {}
```

```
 lvl2: Dict[str, int] = {}
```

```
 file_heap: List[Tuple[int, str]] = []
```

```
 stack = [root]
```

```
 root_parts_len = len(Path(root).parts)
```

```
def add_file(sz: int, p: str):
```

```
 if top_files <= 0:
```

```
 return
```

```
 if len(file_heap) < top_files:
```

```
 heapq.heappush(file_heap, (sz, p))
```

```
 else:
```

```
 if sz > file_heap[0][0]:
```

```
heapq.heapreplace(file_heap, (sz, p))
```

≡ ChatGPT 5.2 Thinking ▾

🔗 ↗ 👤 ...

```
stats.dirs_scanned += 1
try:
 with os.scandir(cur) as it:
 for entry in it:
 try:
 if entry.is_symlink():
 continue
 full = entry.path
 if is_reparse_point(full):
 stats.skipped_reparse += 1
 continue

 if entry.is_dir(follow_symlinks=False):
 rel_parts = len(Path(full).parts) - root_parts_len
 if depth < 0 or rel_parts <= depth:
 stack.append(full)
 elif entry.is_file(follow_symlinks=False):
 st = entry.stat(follow_symlinks=False)
 sz = int(getattr(st, "st_size", 0))
 stats.files_scanned += 1
 stats.bytes_total += sz

 try:
 rel = os.path.relpath(full, root)
 except Exception:
 rel = full
 parts = rel.split(os.sep)
 if len(parts) >= 1:
 lvl1[parts[0]] = lvl1.get(parts[0], 0) + sz
 if len(parts) >= 2:
 k2 = os.path.join(parts[0], parts[1])
 lvl2[k2] = lvl2.get(k2, 0) + sz

 add_file(sz, full)
 except PermissionError:
 stats.denied += 1
 except FileNotFoundError:
 stats.errors += 1
 except OSError:
```

```

 stats.errors += 1
 except PermissionError:
 stats.denied += 1
 except FileNotFoundError:
 stats.errors += 1
 except OSError:
 stats.errors += 1

stats.elapsed_sec = round(time.time() - t0, 2)
top1 = sorted(lvl1.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top2 = sorted(lvl2.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top_files_list = sorted([(p, sz) for (sz, p) in file_heap], key=lambda x: x[1]
return ScanResult(stats=stats, top_dirs_level1=top1, top_dirs_level2=top2, to

def print_drive_table() -> List[Dict[str, str]]:
 rows = []
 print("=== Drives ===")
 for d in list_drives():
 try:
 du = shutil.disk_usage(d)
 used = du.total - du.free
 print(f"{d} Total: {format_gb(du.total)} | Used: {format_gb(used)} |")
 rows.append({"drive": d, "total_gb": f"{du.total/(1024**3):.2f}", "fr
 except Exception as e:
 print(f"{d} (error reading usage: {e})")
 rows.append({"drive": d, "error": str(e)})
 print()
 return rows

@dataclass
class CleanupAction:
 name: str
 description: str
 targets: List[str]
 needs_admin: bool = False

def expand_globs(patterns: List[str]) -> List[Path]:
 out: List[Path] = []
 for pat in patterns:
 pat2 = os.path.expandvars(pat)

```

```

p = Path(pat2)
if "*" in pat2 or "?" in pat2:
 if any(ch in p.name for ch in "*?"):
 parent = p.parent
 if parent.exists():
 out.extend(list(parent.glob(p.name)))
 else:
 out.append(p)
else:
 out.append(p)
uniq, seen = [], set()
for p in out:
 s = str(p).lower()
 if s not in seen:
 uniq.append(p)
 seen.add(s)
return uniq

```

```

def remove_path(p: Path) -> Tuple[bool, str]:
 try:
 if not p.exists():
 return True, "not_found"
 if p.is_symlink():
 return True, "skip_symlink"
 if is_reparse_point(str(p)):
 return True, "skip_reparse_point"
 if p.is_file():
 p.unlink()
 return True, "deleted_file"
 if p.is_dir():
 shutil.rmtree(p, ignore_errors=False)
 return True, "deleted_dir"
 return True, "unknown_type"
 except PermissionError as e:
 return False, f"permission_denied: {e}"
 except Exception as e:
 return False, f"error: {e}"

```

```

def cleanup(actions: List[CleanupAction], yes: bool, outdir: Path) -> Dict[str, o
report = {
 "generated_at": datetime.now().isoformat(timespec="seconds"),

```

```

 "is_admin": is_admin(),
 "mode": "EXECUTE" if yes else "DRY_RUN",
 "actions": [],
}

for act in actions:
 act_entry = {"name": act.name, "description": act.description, "needs_admin":
 if act.needs_admin and not is_admin():
 act_entry["skipped"] = "needs_admin"
 report["actions"].append(act_entry)
 continue

 paths = expand_globs(act.targets)
 for p in paths:
 item = {"path": str(p), "exists": p.exists()}
 if yes:
 ok, msg = remove_path(p)
 item["result"] = msg
 item["ok"] = ok
 act_entry["items"].append(item)

 report["actions"].append(act_entry)

if yes:
 code, out, err = run_powershell("try { Clear-RecycleBin -Force -ErrorActi
 report["recycle_bin"] = {"code": code, "out": out, "err": err}

outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
(outdir / f"cleanup_report_{ts}.json").write_text(json.dumps(report, ensure_a
return report

def winupdate_cache_reset(yes: bool, outdir: Path) -> Dict[str, object]:
 report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "mode":
 report["target"] = os.path.expandvars(r"%windir%\SoftwareDistribution\Downloa

 if not yes:
 report["note"] = "Would stop wuauserv/bits, delete Download cache, then s
 (outdir / f"winupdate_cache_dryrun_{datetime.now().strftime('%Y%m%d_%H%M%
 json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
 return report

```

```

if not is_admin():
 report["error"] = "Run as Administrator for this action."
 (outdir / f"winupdate_cache_error_{datetime.now().strftime('%Y%m%d_%H%M%S')}
 json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
 return report

cmds = [
 ["net", "stop", "wuauserv"],
 ["net", "stop", "bits"],
 ["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-Command",
 r"Remove-Item -Recurse -Force "$env:windir\SoftwareDistribution\Downloa
 ["net", "start", "bits"],
 ["net", "start", "wuauserv"],
]
results = []
for c in cmds:
 code, out, err = run_cmd(c)
 results.append({"cmd": " ".join(c), "code": code, "out": out, "err": err})
report["results"] = results

(outdir / f"winupdate_cache_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json")
 json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
return report

```

```

def backup_browsers(dest: str) -> Dict[str, object]:
 local = Path(os.environ.get("LOCALAPPDATA", ""))
 roam = Path(os.environ.get("APPDATA", ""))
 dest_root = Path(dest).expanduser().resolve()
 stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
 out = dest_root / f"browsers_backup_{stamp}"
 out.mkdir(parents=True, exist_ok=True)

 def copytree_fast(src: Path, dst: Path):
 ignore = shutil.ignore_patterns(
 "Cache", "Code Cache", "GPUCache", "ShaderCache",
 "Media Cache", "GrShaderCache", "DawnCache",
 "Service Worker\\CacheStorage", "Service Worker\\ScriptCache",
 "Crashpad", "Crash Reports"
)

```



```
shutil.copytree(src, dst, dirs_exist_ok=True, ignore=ignore)

sources = {
 "Edge_UserData": local / "Microsoft" / "Edge" / "User Data",
 "Yandex_UserData": local / "Yandex" / "YandexBrowser" / "User Data",
 "Opera_Stable_Roaming": roam / "Opera Software" / "Opera Stable",
 "OperaGX_Stable_Roaming": roam / "Opera Software" / "Opera GX Stable",
}

report = {"saved_to": str(out), "items": []}
for name, src in sources.items():
 if src.exists():
 dst = out / name
 try:
 copytree_fast(src, dst)
 report["items"].append({"name": name, "src": str(src), "dst": str
 except Exception as e:
 report["items"].append({"name": name, "src": str(src), "dst": str
 else:
 report["items"].append({"name": name, "src": str(src), "status": "not

(out / "manifest.json").write_text(json.dumps(report, ensure_ascii=False, ind
return report

def optimize_ssd(drive_letter: str) -> Dict[str, object]:
 report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "driv
 if not is_admin():
 report["error"] = "Run as Administrator for TRIM."
 return report
 ps = f"try {{ Optimize-Volume -DriveLetter {drive_letter} -ReTrim -Verbose }}"
 code, out, err = run_powershell(ps)
 report.update({"code": code, "out": out, "err": err})
 return report

def cmd_scan(args: argparse.Namespace):
 drives_info = print_drive_table()

 roots = args.roots or []
 if not roots:
 user = os.environ.get("USERPROFILE", "")
 roots = [
```

```

 os.path.join(user, "AppData", "Local"),
 os.path.join(user, "AppData", "Roaming"),
 os.path.join(os.environ.get("windir", r"C:\Windows"), "SoftwareDistri
 os.path.join(os.environ.get("windir", r"C:\Windows"), "Temp"),
]

results: List[ScanResult] = []
for r in roots:
 if not r:
 continue
 r = os.path.expandvars(r)
 if not os.path.exists(r):
 print(f"[skip] Not found: {r}")
 continue
 print(f"=== Scanning: {r} (depth={args.depth}) ===")
 res = scan_root(r, depth=args.depth, top_dirs=args.top, top_files=args.fi
 results.append(res)

 st = res.stats
 print(f"Scanned dirs={st.dirs_scanned}, files={st.files_scanned}, total={
 f"denied={st.denied}, errors={st.errors}, skipped_reparse={st.skipp

 print("--- Top folders (level 1) ---")
 for name, sz in res.top_dirs_level1:
 print(f"{name:<45} {format_gb(sz)}")
 print("\n--- Top folders (level 2) ---")
 for name, sz in res.top_dirs_level2:
 print(f"{name:<45} {format_gb(sz)}")
 print("\n--- Top files ---")
 for p, sz in res.top_files:
 print(f"{format_gb(sz):>10} {p}")
 print("\n" + "=" * 70 + "\n")

outdir = Path(args.outdir).resolve()
outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base = outdir / f"scan_report_{ts}"

payload = {
 "generated_at": datetime.now().isoformat(timespec="seconds"),
 "is_admin": is_admin(),
 "drives": drives_info,
 "results": [

```

```

 {"stats": asdict(r.stats), "top_dirs_level1": r.top_dirs_level1, "top.
 for r in results
],
}
(base.with_suffix(".json")).write_text(json.dumps(payload, ensure_ascii=False

with open(base.with_suffix("_topdirs.csv"), "w", encoding="utf-8-sig", newlin
 w = csv.writer(f, delimiter=";")
 w.writerow(["root", "level", "path", "bytes", "gb"])
 for r in results:
 root = r.stats.root
 for p, sz in r.top_dirs_level1:
 w.writerow([root, 1, p, sz, round(sz / (1024**3), 3)])
 for p, sz in r.top_dirs_level2:
 w.writerow([root, 2, p, sz, round(sz / (1024**3), 3)])

print(f"Saved:\n- {base.with_suffix('.json')}\n- {base.with_suffix('_topdirs.

def build_cleanup_actions(include_browser_cache: bool) -> List[CleanupAction]:
 local = r"%LOCALAPPDATA%"
 temp = r"%TEMP%"

 actions = [
 CleanupAction(
 name="Temp (user)",
 description="Clear user TEMP files (safe).",
 targets=[temp + r"*"],
 needs_admin=False,
),
 CleanupAction(
 name="CrashDumps",
 description="Clear Windows Error Reporting crash dumps for user (safe
 targets=[local + r"\CrashDumps*"],
 needs_admin=False,
),
 CleanupAction(
 name="SquirrelTemp",
 description="Clear installer temp used by Squirrel-based updaters (sa
 targets=[local + r"\SquirrelTemp*"],
 needs_admin=False,
),
 CleanupAction(

```

```
name="NVIDIA cache (ProgramData)",
description="Clear NVIDIA downloader + NV_Cache (safe).",
targets=[r"C:\ProgramData\NVIDIA Corporation\Downloader*", r"C:\Prog
needs_admin=True,
),
]

if include_browser_cache:
 actions.extend([
 CleanupAction(
 name="Yandex Browser cache",
 description="Clear Yandex browser caches only (close browser first)",
 targets=[
 local + r"\Yandex\YandexBrowser\User Data*\Cache*",
 local + r"\Yandex\YandexBrowser\User Data*\Code Cache*",
 local + r"\Yandex\YandexBrowser\User Data*\GPUCache*",
 local + r"\Yandex\YandexBrowser\User Data*\Service Worker\Ca
 local + r"\Yandex\YandexBrowser\User Data*\Service Worker\Sc
],
 needs_admin=False,
),
 CleanupAction(
 name="Opera cache",
 description="Clear Opera caches only (close browser first).",
 targets=[
 local + r"\Opera Software\Opera Stable\Cache*",
 local + r"\Opera Software\Opera Stable\Code Cache*",
 local + r"\Opera Software\Opera Stable\GPUCache*",
 local + r"\Opera Software\Opera GX Stable\Cache*",
 local + r"\Opera Software\Opera GX Stable\Code Cache*",
 local + r"\Opera Software\Opera GX Stable\GPUCache*",
],
 needs_admin=False,
),
 CleanupAction(
 name="Edge cache",
 description="Clear Edge caches only (close browser first).",
 targets=[
 local + r"\Microsoft\Edge\User Data*\Cache*",
 local + r"\Microsoft\Edge\User Data*\Code Cache*",
 local + r"\Microsoft\Edge\User Data*\GPUCache*",
 local + r"\Microsoft\Edge\User Data*\Service Worker\CacheSto
 local + r"\Microsoft\Edge\User Data*\Service Worker\ScriptCa
```

```
],
 needs_admin=False,
),
])

return actions

def cmd_cleanup(args: argparse.Namespace):
 outdir = Path(args.outdir).resolve()
 actions = build_cleanup_actions(include_browser_cache=args.browser_cache)

 print(f"Mode: {'EXECUTE' if args.yes else 'DRY_RUN'}")
 if args.browser_cache:
 print("Including browser caches: YES (close browsers before executing)")

 rep = cleanup(actions, yes=args.yes, outdir=outdir)
 print(f"Report saved into: {outdir}")
 if rep.get("mode") == "DRY_RUN":
 print("Dry-run complete. Re-run with --yes to actually delete.")
 else:
 print("Cleanup executed.")

def cmd_backup(args: argparse.Namespace):
 rep = backup_browsers(args.dest)
 print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_winupdate_cache(args: argparse.Namespace):
 outdir = Path(args.outdir).resolve()
 rep = winupdate_cache_reset(yes=args.yes, outdir=outdir)
 print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_trim(args: argparse.Namespace):
 rep = optimize_ssd(args.drive)
 print(json.dumps(rep, ensure_ascii=False, indent=2))

def main():
 if os.name != "nt":
 print("This tool is for Windows.")
```

```

sys.exit(1)

ap = argparse.ArgumentParser("win_maintain", description="Scan + safe cleanup
ap.add_argument("--outdir", default=".", help="Output dir for reports (default:

sub = ap.add_subparsers(dest="cmd", required=True)

sp_scan = sub.add_parser("scan", help="Scan disk hogs under common roots (or
sp_scan.add_argument("--roots", nargs="*", default=[], help="Roots to scan (d
sp_scan.add_argument("--depth", type=int, default=6, help="Relative depth (de
sp_scan.add_argument("--top", type=int, default=25, help="Top N folders (defa
sp_scan.add_argument("--files", type=int, default=30, help="Top N files (defa
sp_scan.set_defaults(func=cmd_scan)

sp_cleanup = sub.add_parser("cleanup", help="Safe cleanup (temp/caches). Defa
sp_cleanup.add_argument("--yes", action="store_true", help="Actually delete (
sp_cleanup.add_argument("--browser-cache", action="store_true", help="Also cl
sp_cleanup.set_defaults(func=cmd_cleanup)

sp_backup = sub.add_parser("backup-browsers", help="Backup browser profiles t
sp_backup.add_argument("--dest", default=r"D:\Backups\Browsers", help="Destin
sp_backup.set_defaults(func=cmd_backup)

sp_wu = sub.add_parser("winupdate-cache", help="Reset Windows Update download
sp_wu.add_argument("--yes", action="store_true", help="Execute (otherwise dry
sp_wu.set_defaults(func=cmd_winupdate_cache)

sp_trim = sub.add_parser("trim", help="Run SSD TRIM (Optimize-Volume) for a d
sp_trim.add_argument("--drive", default="C", help="Drive letter (default C).
sp_trim.set_defaults(func=cmd_trim)

args = ap.parse_args()
args.func(args)

if __name__ == "__main__":
 main()
...

readme = r'''# win_maintain_pack

```

Набор скриптов для Windows 10/11: анализ «что жрёт место», безопасная очистка кэш

### ## Содержимое

- `win\_maintain.py` — основной «комбайн»:
- `scan` — сканирует выбранные корни и показывает ТОП папок/файлов
- `cleanup` — безопасная очистка (по умолчанию dry-run)
- `cleanup --browser-cache` — чистит только кэши браузеров (профили/пароли/исто
- `backup-browsers` — бэкап профилей Edge/Yandex/Opera (без тяжёлых кэшей)
- `winupdate-cache` — очистка кэша скачанных обновлений Windows Update
- `trim` — TRIM для SSD (Optimize-Volume)

### ## Запуск

```
```powershell
```

```
python .\win_maintain.py scan --outdir .
python .\win_maintain.py cleanup --outdir .          # dry-run
python .\win_maintain.py cleanup --yes --outdir .    # выполнить
python .\win_maintain.py cleanup --yes --browser-cache --outdir .
python .\win_maintain.py backup-browsers --dest D:\Backups\Browsers
python .\win_maintain.py winupdate-cache --yes --outdir .
python .\win_maintain.py trim --drive C
```

Важно

- Для `cleanup --browser-cache` **закрой** браузеры перед запуском (иначе часть файлов будет занята).
 - Скрипт **не удаляет** профили браузеров и не трогает пароли/куки/историю (чистит только Cache/Code Cache/GPUCache/Service Worker CacheStorage).
 - Папки `AppData\Local\Programs` , `AppData\Local\Packages` , `AppData\Local\Microsoft` — это не «мусор». Их надо чистить точно. Скрипт их не трогает.
- ...

```
with open(os.path.join(base_dir, "win_maintain.py"), "w", encoding="utf-8") as f:
f.write(win_maintain_py)
with open(os.path.join(base_dir, "README.md"), "w", encoding="utf-8") as f:
f.write(readme)
```

```
zip_path = "/mnt/data/win_maintain_pack.zip"
with zipfile.ZipFile(zip_path, "w", compression=zipfile.ZIP_DEFLATED) as z:
z.write(os.path.join(base_dir, "win_maintain.py"), arcname="win_maintain.py")
z.write(os.path.join(base_dir, "README.md"), arcname="README.md")
```

```
zip_path
```

Результат

```
'/mnt/data/win_maintain_pack.zip'
```

Вижу твою статистику — всё идеально читается. И да: **ты уже молодец** — не «удалил Windows», а почистил по уму 😊

Что у тебя реально “жрёт” в AppData (и что с этим делать)

Local (самые толстые)

1. Local\Yandex (3.11 GB)

Это почти наверняка кэш Яндекс.Браузера (Cache / Code Cache / GPU Cache / Service Worker).

✅ Можно чистить безопасно, если чистишь именно кэши, а не весь профиль.

2. Local\Programs (2.48 GB)

Это установленные программы “на пользователя” (Python, всякие апдейтеры, возможно Discord/Telegram-версии и т.д.).

⚠️ Это НЕ мусор. Удалять руками — значит “удалить программу по-живому”.
Правильно: **удаление через “Приложения и возможности”** или переустановка на D:.

3. Local\Microsoft (2.43 GB)

Тут может быть Edge/WebView2/Teams/Office-кэш.

⚠️ Ручная чистка “всё подряд” не советуется. Только точно (кэши), или через настройки приложений.

4. Local\Packages (1.92 GB)

Это Microsoft Store приложения.

⚠️ Ручное удаление там — лотерея. Лучше чистить через:

- Параметры → Приложения → Приложения и возможности → конкретное приложение → **Сброс/Восстановление**
- или “Хранилище”/Storage Sense

5. Local\JetBrains (1.2 GB) + Roaming\JetBrains (1.24 GB)

Там часто кэши, индексы, логи.

✅ Можно чистить кэши (IDE потом переиндексирует проекты и первые 5–15 минут будет “думать”).

Лучше вариант на будущее: **перенос кэша JetBrains на D:** (скажу как, если надо).

6. Local\GitHubDesktop (1.01 GB)

Обычно кэш/логи.

✓ можно чистить, но аккуратно (лучше через настройки/переустановку, или чистить Cache/Logs).

7. Local\SquirrelTemp (0.65 GB)

✓ Абсолютно безопасный мусор. Можно удалять хоть каждый день.

8. Opera (Local 0.45 + Roaming 0.56)

Roaming — профиль (ценный), Local — часть кэшей.

✓ чистить можно **только** Cache/Code Cache/GPUCache (профиль не трогаем).

Ты уже сделал правильные команды

- `powercfg -h off` ✓ (убил hiberfil.sys — обычно несколько ГБ)
- `DISM /StartComponentCleanup` ✓
- чистка Windows Update Download ✓
- TEMP + корзина ✓

И результат: C: free = 30.69GB — это прям “винда задышала”.

Дальше: “умная чистка” без потери паролей/истории/сессий

Твой “топ безопасных целей”:

- SquirrelTemp
- CrashDumps
- %TEMP%
- `pip cache purge`
- **кэши браузеров** (Cache/Code Cache/GPUCache/Service Worker CacheStorage) — но закрыть браузер перед этим.

Комбайн-скрипт (анализ + очистка + бэкап браузеров)

Я собрал тебе **единый пакет**:

- анализ (scan) — покажет что и где жрёт, + сохранит отчёт JSON/CSV
- очистка (cleanup) — *по умолчанию dry-run*

- очистка кэшей браузеров (cleanup --browser-cache) — **профили/пароли не трогает**
- бэкап браузеров (backup-browsers) — копирует профили Edge/Yandex/Opera **без тяжёлых кэшей**
- очистка WinUpdate cache (winupdate-cache)
- TRIM SSD (trim)

✅ Вот архив:

[Download win_maintain_pack.zip](#)

Как запускать у себя

Распакуй, открой PowerShell в папке, и:

```
powershell
```

 Копировать код

```
python .\win_maintain.py scan --outdir .
```

Очистка (сначала безопасный просмотр):

```
powershell
```

 Копировать код

```
python .\win_maintain.py cleanup --outdir .
```

Реально удалить (выполнить):

```
powershell
```

 Копировать код

```
python .\win_maintain.py cleanup --yes --outdir .
```

Почистить кэши браузеров (закрой браузеры!):

```
powershell
```

 Копировать код

```
python .\win_maintain.py cleanup --yes --browser-cache --outdir .
```

Сделать бэкап профилей браузеров:

```
powershell
```

 Копировать код

```
python .\win_maintain.py backup-browsers --dest D:\Backups\Browsers
```

Чтобы я “докрутил под тебя”

Скажи 2 вещи:

1. какой браузер основной: Yandex / Opera / Edge?
2. JetBrains — это PyCharm/IntelliJ и какая версия примерно?

И я добавлю в пакет “умные правила” именно под твой набор:

- JetBrains: чистка только кэшей/логов + (опционально) перенос system/cache на D
- Windsurf/Cursor: чистка логов/кэшей без убийства настроек
- Telegram Desktop: чистка кэша (или хотя бы напомним где лимит ставится)

А пока — у тебя уже состояние: **Win10 можно спокойно оставить и довести до “летает”**. </>



Write files

```
with open(os.path.join(base_dir, "win_maintain.py"), "w", encoding="utf-8") as f:
    f.write(win_maintain.py)
with open(os.path.join(base_dir, "README.md"), "w", encoding="utf-8") as f:
    f.write(readme)

zip_path = "/mnt/data/win_maintain_pack.zip"
with zipfile.ZipFile(zip_path, "w", compression=zipfile.ZIP_DEFLATED) as z:
    for fn in ["win_maintain.py", "README.md"]:
        z.write(os.path.join(base_dir, fn), arcname=fn)
```

zip_path

Всегда показывать подробности

Копировать код

Анализ

python

Всегда показывать подробности

Копировать код

что это за кусок?

и вот еще кстати что выполнял и получил

Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Windows\system32> $root =
"C:\ProgramData\NVIDIA Corporation\NVIDIA app"
>> Get-ChildItem $root -Directory -Force -ErrorAction
SilentlyContinue |
>>   ForEach-Object {
>>     $sum = (Get-ChildItem $_.FullName -File -Recurse -
Force -ErrorAction SilentlyContinue |
>>       Measure-Object Length -Sum).Sum
>>     [pscustomobject]@{ Path=$_.FullName; GB=
```

```
[math]::Round($sum/1GB,2) }
>> } | Sort-Object GB -Descending | Select-Object -First
30
>>
```

Path	GB
C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework	3.56
C:\ProgramData\NVIDIA Corporation\NVIDIA app\NvDriverDiagnostics	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\NvContainer	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\UXD	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\SessionLogs	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\Logs	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\Installer	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\MessageBus	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\NvConfig	0
C:\ProgramData\NVIDIA Corporation\NVIDIA app\NvBackend	0

```
PS C:\Windows\system32> Remove-Item -Recurse -Force
"$env:LOCALAPPDATA\JetBrains\IdealC2024.3\caches" -
ErrorAction SilentlyContinue
>> Remove-Item -Recurse -Force
"$env:LOCALAPPDATA\JetBrains\IdealC2024.3\log" -
ErrorAction SilentlyContinue
>>
PS C:\Windows\system32> [math]::Round((Get-PSDrive
C).Free/1GB,2)
>>
31.42
PS C:\Windows\system32>
```

Думал на протяжении 41s >

Это нормальный вопрос — на скрине выглядит как “что за магия, я это не запускал”.

1) “Что это за кусок?”

Это служебный код из среды ChatGPT, который собирает архив с файлами (win_maintain.py и README.md) и кладёт его в win_maintain_pack.zip.

То есть:

- на твоём ПК он не выполнялся
- это просто “упаковщик” в моём окружении, чтобы тебе отдать готовый zip
- ты у себя запускаешь **только** win_maintain.py (или распакованный архив), а не этот кусок.

Если в интерфейсе включено “показывать подробности”, такие фрагменты иногда видны — это не вредняк и не “команда на твою винду”.

2) По твоим новым результатам — всё отлично

JetBrains

Ты сделал:

```
powershell
```

 Копировать код

```
Remove-Item ...\\JetBrains\\IdeaIC2024.3\\caches  
Remove-Item ...\\JetBrains\\IdeaIC2024.3\\log
```

Это **нормально и безопасно**. IDE просто заново построит кэши/индексы (первый запуск будет чуть “думать”).

И ты реально выиграл место:

Free C: 31.42 GB 

3) NVIDIA app: UpdateFramework 3.56 GB — что это и можно ли чистить?

C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework — ЭТО ПОЧТИ ВСЕГДА кэш обновлятора/диагностики/скачанных пакетов NVIDIA app.

✅ В большинстве случаев чистить можно, максимум — NVIDIA app потом заново скачает нужное.

Сначала — глянь, что именно там лежит (ТОП-файлы)

powershell

Копировать код

```
$root = "C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework"
Get-ChildItem $root -Recurse -File -Force -ErrorAction SilentlyContinue |
    Sort-Object Length -Descending |
    Select-Object -First 30 FullName, @{n="GB";e={[math]::Round($_.Length/1GB,2)}}
```

Если увидишь там какие-то .exe/.msi/.zip на гигабайты — это оно.

Потом — безопасная чистка (лучше PowerShell от администратора)

Закрой NVIDIA app и подчисти кэш:

powershell

Копировать код

```
# Закрыть NVIDIA app (если запущено)
Get-Process -ErrorAction SilentlyContinue | Where-Object {$_.ProcessName -like "*

# Удалить содержимое UpdateFramework
Remove-Item -Recurse -Force "C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateF

# (опционально) если папка Installer тоже раздута – можно так же
Remove-Item -Recurse -Force "C:\ProgramData\NVIDIA Corporation\NVIDIA app\Install
```

Если что-то “занято системой” — просто перезагрузи ПК и повтори.

4) Хочешь — добавлю NVIDIA UpdateFramework в твой “комбайн”-скрипт

В win_maintain.py можно добавить ещё одно действие в build_cleanup_actions() :

python

Копировать код

```
CleanupAction(
    name="NVIDIA app UpdateFramework",
    description="Clear NVIDIA app UpdateFramework cache (will re-download if need
    targets=[
        r"C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework\*",
        r"C:\ProgramData\NVIDIA Corporation\NVIDIA app\Installer\*",
    ],
    needs_admin=True,
),
```

Скажи — ты пользуешься **NVIDIA app** активно (оверлей/запись/обновления), или она тебе просто “поставилась и живёт”?

Если она не нужна — иногда проще **удалить NVIDIA app**, а драйвер оставить (но это уже по твоему комфорту).

Если кинешь вывод “ТОП-файлов” из UpdateFramework (30 строк), я скажу “вот это можно сносить смело, а вот это лучше оставить”.



нет не особо пользуюсь

вот вывод

Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (<https://aka.ms/pscore6>)

```
PS C:\Windows\system32> $root =
"C:\ProgramData\NVIDIA Corporation\NVIDIA
app\UpdateFramework"
>> Get-ChildItem $root -Recurse -File -Force -ErrorAction
SilentlyContinue |
>> Sort-Object Length -Descending |
>> Select-Object -First 30 FullName, @{n="GB";e=
{[math]::Round($_.Length/1GB,2)}}
>>
```

• •

C:\ProgramData\NVIDIA Corporation\NVIDIA

[illegible]

```
processing\591d2fefff406d921c4c1...
C:\ProgramData\NVIDIA Corporation\NVIDIA
app\UpdateFramework\ota-artifacts\grd\post-
processing\591d2fefff406d921c4c1...
C:\ProgramData\NVIDIA Corporation\NVIDIA
app\UpdateFramework\ota-artifacts\grd\post-
processing\591d2fefff406d921c4c1...
```

PS C:\Windows\system32>

но кстати

запиши всю историю, скрипты, историю запросов
результатов и историю чатов и тд в лог или несколько
файлов или архив

подготовь скрипты для очистки

напиши промт мб даже для другой беседы чтобы
интегрировать или объединить логику

тот же чат гпт

только в другом чате (беседе)

постараюсь сделать одну сборку файлов скриптов

запусков батников и тд

и по сути все что мы делали

по шагам или вместе

для этого компа например и для ноутбука, он по моему
на винд 11

Думал на протяжении 58s >

Анализ ошибочен ^

python

```
import os, zipfile, textwrap, js
from datetime import datetime
```

Всегда показывать подробности

 Копировать код

```
base_dir = "/mnt/data/win_maintain_pack_v2"
os.makedirs(base_dir, exist_ok=True)
```

```
win_maintain_v2 = r'''# win_maintain.py (v2)
# Windows 10/11 maintenance helper: scan disk hogs, backup browser profiles, and
```

```
# No external deps. Python 3.9+ recommended.
#
# Safety:
# - Default cleanup mode is DRY-RUN (shows what would be deleted).
# - Only deletes known cache/temp locations unless you explicitly enable extra fl
# - Skips symlinks/reparse points while scanning to avoid loops.
#
# Usage examples:
# python win_maintain.py scan
# python win_maintain.py cleanup                (dry-run)
# python win_maintain.py cleanup --yes          (execute)
# python win_maintain.py cleanup --yes --browser-cache
# python win_maintain.py cleanup --yes --nvidia-app-cache
# python win_maintain.py backup-browsers --dest D:\Backups\Browsers
# python win_maintain.py winupdate-cache --yes
#
# Tip: run PowerShell as Administrator for best results.
```

```
from __future__ import annotations
```

```
import argparse
import csv
import ctypes
import heapq
import json
import os
import shutil
import subprocess
import sys
import time
from dataclasses import dataclass, asdict
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Tuple
```

```
REPARSE_POINT_ATTR = 0x0400 # FILE_ATTRIBUTE_REPARSE_POINT
```

```
def is_admin() -> bool:
    try:
        return bool(ctypes.windll.shell32.IsUserAnAdmin())
    except Exception:
```

```
        return False

def decode_best_effort(b: bytes) -> str:
    if b is None:
        return ""
    for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
        try:
            return b.decode(enc)
        except Exception:
            continue
    return b.decode("utf-8", errors="replace")

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
    p = subprocess.run(cmd, capture_output=True)
    out = decode_best_effort(p.stdout).strip()
    err = decode_best_effort(p.stderr).strip()
    return p.returncode, out, err

def run_powershell(ps: str) -> Tuple[int, str, str]:
    prefix = "[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new(); $Pro"
    return run_cmd(["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-C"

def format_gb(num_bytes: float) -> str:
    return f"{num_bytes / (1024**3):.2f} GB"

def is_reparse_point(path: str) -> bool:
    try:
        st = os.stat(path, follow_symlinks=False)
        attrs = getattr(st, "st_file_attributes", 0)
        return bool(attrs & REPARSE_POINT_ATTR)
    except Exception:
        return False

def list_drives() -> List[str]:
    drives = []
    for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        p = f"{c}:\\"

```

```
        if os.path.exists(p):
            drives.append(p)
    return drives
```

```
@dataclass
```

```
class ScanStats:
```

```
    root: str
    depth: int
    files_scanned: int = 0
    dirs_scanned: int = 0
    bytes_total: int = 0
    errors: int = 0
    denied: int = 0
    skipped_reparse: int = 0
    elapsed_sec: float = 0.0
```

```
@dataclass
```

```
class ScanResult:
```

```
    stats: ScanStats
    top_dirs_level1: List[Tuple[str, int]]
    top_dirs_level2: List[Tuple[str, int]]
    top_files: List[Tuple[str, int]]
```

```
def scan_root(root: str, depth: int, top_dirs: int, top_files: int) -> ScanResult
```

```
    t0 = time.time()
    root = os.path.abspath(root)
    stats = ScanStats(root=root, depth=depth)
```

```
    lvl1: Dict[str, int] = {}
    lvl2: Dict[str, int] = {}
    file_heap: List[Tuple[int, str]] = []
```

```
    stack = [root]
    root_parts_len = len(Path(root).parts)
```

```
    def add_file(sz: int, p: str):
        if top_files <= 0:
            return
        if len(file_heap) < top_files:
            heapq.heappush(file_heap, (sz, p))
```

```
else:
    if sz > file_heap[0][0]:
        heapq.heapreplace(file_heap, (sz, p))

while stack:
    cur = stack.pop()
    stats.dirs_scanned += 1
    try:
        with os.scandir(cur) as it:
            for entry in it:
                try:
                    if entry.is_symlink():
                        continue
                    full = entry.path
                    if is_reparse_point(full):
                        stats.skipped_reparse += 1
                        continue

                    if entry.is_dir(follow_symlinks=False):
                        rel_parts = len(Path(full).parts) - root_parts_len
                        if depth < 0 or rel_parts <= depth:
                            stack.append(full)
                    elif entry.is_file(follow_symlinks=False):
                        st = entry.stat(follow_symlinks=False)
                        sz = int(getattr(st, "st_size", 0))
                        stats.files_scanned += 1
                        stats.bytes_total += sz

                        try:
                            rel = os.path.relpath(full, root)
                        except Exception:
                            rel = full
                        parts = rel.split(os.sep)
                        if len(parts) >= 1:
                            lvl1[parts[0]] = lvl1.get(parts[0], 0) + sz
                        if len(parts) >= 2:
                            k2 = os.path.join(parts[0], parts[1])
                            lvl2[k2] = lvl2.get(k2, 0) + sz

                    add_file(sz, full)
                except PermissionError:
                    stats.denied += 1
                except FileNotFoundError:
```

```

        stats.errors += 1
    except OSError:
        stats.errors += 1
except PermissionError:
    stats.denied += 1
except FileNotFoundError:
    stats.errors += 1
except OSError:
    stats.errors += 1

stats.elapsed_sec = round(time.time() - t0, 2)
top1 = sorted(lvl1.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top2 = sorted(lvl2.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top_files_list = sorted([(p, sz) for (sz, p) in file_heap], key=lambda x: x[1]
return ScanResult(stats=stats, top_dirs_level1=top1, top_dirs_level2=top2, to

def print_drive_table() -> List[Dict[str, str]]:
    rows = []
    print("=== Drives ===")
    for d in list_drives():
        try:
            du = shutil.disk_usage(d)
            used = du.total - du.free
            print(f"{d} Total: {format_gb(du.total)} | Used: {format_gb(used)} |")
            rows.append({"drive": d, "total_gb": f"{du.total/(1024**3):.2f}", "fr
        except Exception as e:
            print(f"{d} (error reading usage: {e})")
            rows.append({"drive": d, "error": str(e)})
    print()
    return rows

@dataclass
class CleanupAction:
    name: str
    description: str
    targets: List[str]
    needs_admin: bool = False

def expand_globs(patterns: List[str]) -> List[Path]:
    out: List[Path] = []

```

```

for pat in patterns:
    pat2 = os.path.expandvars(pat)
    p = Path(pat2)
    if "*" in pat2 or "?" in pat2:
        parent = p.parent
        if parent.exists():
            out.extend(list(parent.glob(p.name)))
        else:
            out.append(p)
    else:
        out.append(p)
uniq, seen = [], set()
for p in out:
    s = str(p).lower()
    if s not in seen:
        uniq.append(p)
        seen.add(s)
return uniq

def remove_path(p: Path) -> Tuple[bool, str]:
    try:
        if not p.exists():
            return True, "not_found"
        if p.is_symlink():
            return True, "skip_symlink"
        if is_reparse_point(str(p)):
            return True, "skip_reparse_point"
        if p.is_file():
            p.unlink()
            return True, "deleted_file"
        if p.is_dir():
            shutil.rmtree(p, ignore_errors=False)
            return True, "deleted_dir"
        return True, "unknown_type"
    except PermissionError as e:
        return False, f"permission_denied: {e}"
    except Exception as e:
        return False, f"error: {e}"

def cleanup(actions: List[CleanupAction], yes: bool, outdir: Path) -> Dict[str, o
    report = {

```



```

    "generated_at": datetime.now().isoformat(timespec="seconds"),
    "is_admin": is_admin(),
    "mode": "EXECUTE" if yes else "DRY_RUN",
    "actions": [],
}

for act in actions:
    act_entry = {"name": act.name, "description": act.description, "needs_admin": act.needs_admin}
    if act.needs_admin and not is_admin():
        act_entry["skipped"] = "needs_admin"
        report["actions"].append(act_entry)
        continue

    paths = expand_globs(act.targets)
    for p in paths:
        item = {"path": str(p), "exists": p.exists()}
        if yes:
            ok, msg = remove_path(p)
            item["result"] = msg
            item["ok"] = ok
            act_entry["items"].append(item)

    report["actions"].append(act_entry)

if yes:
    code, out, err = run_powershell("try { Clear-RecycleBin -Force -ErrorAction SilentlyContinue }")
    report["recycle_bin"] = {"code": code, "out": out, "err": err}

outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
(outdir / f"cleanup_report_{ts}.json").write_text(json.dumps(report, ensure_ascii=False))
return report

def winupdate_cache_reset(yes: bool, outdir: Path) -> Dict[str, object]:
    report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "mode": "EXECUTE" if yes else "DRY_RUN"}
    report["target"] = os.path.expandvars(r"%windir%\SoftwareDistribution\Download")

    if not yes:
        report["note"] = "Would stop wuauserv/bits, delete Download cache, then stop wuauserv/bits"
        (outdir / f"winupdate_cache_dryrun_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json").write_text(
            json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
        )

```

```

        return report

    if not is_admin():
        report["error"] = "Run as Administrator for this action."
        (outdir / f"winupdate_cache_error_{datetime.now().strftime('%Y%m%d_%H%M%S')}").json.dumps(
            report, ensure_ascii=False, indent=2, encoding="utf-8"
        )
        return report

    cmds = [
        ["net", "stop", "wuauserv"],
        ["net", "stop", "bits"],
        ["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-Command",
         r"Remove-Item -Recurse -Force "$env:windir\SoftwareDistribution\Download"],
        ["net", "start", "bits"],
        ["net", "start", "wuauserv"],
    ]
    results = []
    for c in cmds:
        code, out, err = run_cmd(c)
        results.append({"cmd": " ".join(c), "code": code, "out": out, "err": err})
    report["results"] = results

    (outdir / f"winupdate_cache_{datetime.now().strftime('%Y%m%d_%H%M%S')}").json.dumps(
        report, ensure_ascii=False, indent=2, encoding="utf-8"
    )
    return report


def backup_browsers(dest: str) -> Dict[str, object]:
    local = Path(os.environ.get("LOCALAPPDATA", ""))
    roam = Path(os.environ.get("APPDATA", ""))
    dest_root = Path(dest).expanduser().resolve()
    stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    out = dest_root / f"browsers_backup_{stamp}"
    out.mkdir(parents=True, exist_ok=True)

    def copytree_fast(src: Path, dst: Path):
        ignore = shutil.ignore_patterns(
            "Cache", "Code Cache", "GPUCache", "ShaderCache",
            "Media Cache", "GrShaderCache", "DawnCache",
            "Service Worker\\CacheStorage", "Service Worker\\ScriptCache",
            "Crashpad", "Crash Reports"

```

```

    )
    shutil.copytree(src, dst, dirs_exist_ok=True, ignore=ignore)

sources = {
    "Edge_UserData": local / "Microsoft" / "Edge" / "User Data",
    "Yandex_UserData": local / "Yandex" / "YandexBrowser" / "User Data",
    "Opera_Stable_Roaming": roam / "Opera Software" / "Opera Stable",
    "OperaGX_Stable_Roaming": roam / "Opera Software" / "Opera GX Stable",
}

report = {"saved_to": str(out), "items": []}
for name, src in sources.items():
    if src.exists():
        dst = out / name
        try:
            copytree_fast(src, dst)
            report["items"].append({"name": name, "src": str(src), "dst": str
except Exception as e:
            report["items"].append({"name": name, "src": str(src), "dst": str
        else:
            report["items"].append({"name": name, "src": str(src), "status": "not

(out / "manifest.json").write_text(json.dumps(report, ensure_ascii=False, ind
return report

def optimize_ssd(drive_letter: str) -> Dict[str, object]:
    report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "driv
    if not is_admin():
        report["error"] = "Run as Administrator for TRIM."
        return report
    ps = f"try {{ Optimize-Volume -DriveLetter {drive_letter} -ReTrim -Verbose }}"
    code, out, err = run_powershell(ps)
    report.update({"code": code, "out": out, "err": err})
    return report

def cmd_scan(args: argparse.Namespace):
    drives_info = print_drive_table()

    roots = args.roots or []
    if not roots:
        user = os.environ.get("USERPROFILE", "")

```

```

roots = [
    os.path.join(user, "AppData", "Local"),
    os.path.join(user, "AppData", "Roaming"),
    os.path.join(os.environ.get("windir", r"C:\Windows"), "SoftwareDistri
    os.path.join(os.environ.get("windir", r"C:\Windows"), "Temp"),
]

results: List[ScanResult] = []
for r in roots:
    if not r:
        continue
    r = os.path.expandvars(r)
    if not os.path.exists(r):
        print(f"[skip] Not found: {r}")
        continue
    print(f"=== Scanning: {r} (depth={args.depth}) ===")
    res = scan_root(r, depth=args.depth, top_dirs=args.top, top_files=args.fi
    results.append(res)

    st = res.stats
    print(f"Scanned dirs={st.dirs_scanned}, files={st.files_scanned}, total={
        f"denied={st.denied}, errors={st.errors}, skipped_reparse={st.skipp

    print("--- Top folders (level 1) ---")
    for name, sz in res.top_dirs_level1:
        print(f"{name:<45} {format_gb(sz)}")
    print("\n--- Top folders (level 2) ---")
    for name, sz in res.top_dirs_level2:
        print(f"{name:<45} {format_gb(sz)}")
    print("\n--- Top files ---")
    for p, sz in res.top_files:
        print(f"{format_gb(sz):>10} {p}")
    print("\n" + "=" * 70 + "\n")

outdir = Path(args.outdir).resolve()
outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base = outdir / f"scan_report_{ts}"

payload = {
    "generated_at": datetime.now().isoformat(timespec="seconds"),
    "is_admin": is_admin(),
    "drives": drives_info,

```

```

    "results": [
        {"stats": asdict(r.stats), "top_dirs_level1": r.top_dirs_level1, "top.
        for r in results
    ],
}
(base.with_suffix(".json")).write_text(json.dumps(payload, ensure_ascii=False

with open(base.with_suffix("_topdirs.csv"), "w", encoding="utf-8-sig", newlin
    w = csv.writer(f, delimiter=";")
    w.writerow(["root", "level", "path", "bytes", "gb"])
    for r in results:
        root = r.stats.root
        for p, sz in r.top_dirs_level1:
            w.writerow([root, 1, p, sz, round(sz / (1024**3), 3)])
        for p, sz in r.top_dirs_level2:
            w.writerow([root, 2, p, sz, round(sz / (1024**3), 3)])

print(f"Saved:\n- {base.with_suffix('.json')}\n- {base.with_suffix('_topdirs.

def build_cleanup_actions(include_browser_cache: bool, include_nvidia_app: bool)
    local = r"%LOCALAPPDATA%"
    temp = r"%TEMP%"

    actions = [
        CleanupAction(
            name="Temp (user)",
            description="Clear user TEMP files (safe).",
            targets=[temp + r"\*"],
            needs_admin=False,
        ),
        CleanupAction(
            name="CrashDumps",
            description="Clear Windows Error Reporting crash dumps for user (safe
            targets=[local + r"\CrashDumps\*"],
            needs_admin=False,
        ),
        CleanupAction(
            name="SquirrelTemp",
            description="Clear installer temp used by Squirrel-based updaters (sa
            targets=[local + r"\SquirrelTemp\*"],
            needs_admin=False,
        ),
    ],

```

```

CleanupAction(
    name="NVIDIA cache (ProgramData)",
    description="Clear NVIDIA downloader + NV_Cache (safe).",
    targets=[r"C:\ProgramData\NVIDIA Corporation\Downloader\*", r"C:\Prog
needs_admin=True,
),
]

if include_nvidia_app:
    actions.append(
        CleanupAction(
            name="NVIDIA app UpdateFramework",
            description="Clear NVIDIA app UpdateFramework cache (will re-down
targets=[
            r"C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework
            r"C:\ProgramData\NVIDIA Corporation\NVIDIA app\Installer\*",
        ],
        needs_admin=True,
    )
)

if include_browser_cache:
    actions.extend([
        CleanupAction(
            name="Yandex Browser cache",
            description="Clear Yandex browser caches only (close browser first
targets=[
            local + r"\Yandex\YandexBrowser\User Data\*\Cache\*",
            local + r"\Yandex\YandexBrowser\User Data\*\Code Cache\*",
            local + r"\Yandex\YandexBrowser\User Data\*\GPUCache\*",
            local + r"\Yandex\YandexBrowser\User Data\*\Service Worker\Ca
            local + r"\Yandex\YandexBrowser\User Data\*\Service Worker\Sc
        ],
        needs_admin=False,
    ),
        CleanupAction(
            name="Opera cache",
            description="Clear Opera caches only (close browser first).",
            targets=[
                local + r"\Opera Software\Opera Stable\Cache\*",
                local + r"\Opera Software\Opera Stable\Code Cache\*",
                local + r"\Opera Software\Opera Stable\GPUCache\*",
                local + r"\Opera Software\Opera GX Stable\Cache\*",
            ]
        )
    ])

```

```

        local + r"\Opera Software\Opera GX Stable\Code Cache\*",
        local + r"\Opera Software\Opera GX Stable\GPUCache\*",
    ],
    needs_admin=False,
),
CleanupAction(
    name="Edge cache",
    description="Clear Edge caches only (close browser first).",
    targets=[
        local + r"\Microsoft\Edge\User Data\*\Cache\*",
        local + r"\Microsoft\Edge\User Data\*\Code Cache\*",
        local + r"\Microsoft\Edge\User Data\*\GPUCache\*",
        local + r"\Microsoft\Edge\User Data\*\Service Worker\CacheSto
        local + r"\Microsoft\Edge\User Data\*\Service Worker\ScriptCa
    ],
    needs_admin=False,
),
])

return actions

def cmd_cleanup(args: argparse.Namespace):
    outdir = Path(args.outdir).resolve()
    actions = build_cleanup_actions(
        include_browser_cache=args.browser_cache,
        include_nvidia_app=args.nvidia_app_cache
    )

    print(f"Mode: {'EXECUTE' if args.yes else 'DRY_RUN'}")
    if args.browser_cache:
        print("Including browser caches: YES (close browsers before executing)")
    if args.nvidia_app_cache:
        print("Including NVIDIA app UpdateFramework cache: YES (admin + NVIDIA ap

    rep = cleanup(actions, yes=args.yes, outdir=outdir)
    print(f"Report saved into: {outdir}")
    if rep.get("mode") == "DRY_RUN":
        print("Dry-run complete. Re-run with --yes to actually delete.")
    else:
        print("Cleanup executed.")

```

```
def cmd_backup(args: argparse.Namespace):
    rep = backup_browsers(args.dest)
    print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_winupdate_cache(args: argparse.Namespace):
    outdir = Path(args.outdir).resolve()
    rep = winupdate_cache_reset(yes=args.yes, outdir=outdir)
    print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_trim(args: argparse.Namespace):
    rep = optimize_ssd(args.drive)
    print(json.dumps(rep, ensure_ascii=False, indent=2))

def main():
    if os.name != "nt":
        print("This tool is for Windows.")
        sys.exit(1)

    ap = argparse.ArgumentParser("win_maintain", description="Scan + safe cleanup")
    ap.add_argument("--outdir", default=".", help="Output dir for reports (default: .)")

    sub = ap.add_subparsers(dest="cmd", required=True)

    sp_scan = sub.add_parser("scan", help="Scan disk hogs under common roots (or")
    sp_scan.add_argument("--roots", nargs="*", default=[], help="Roots to scan (default: .)")
    sp_scan.add_argument("--depth", type=int, default=6, help="Relative depth (default: 6)")
    sp_scan.add_argument("--top", type=int, default=25, help="Top N folders (default: 25)")
    sp_scan.add_argument("--files", type=int, default=30, help="Top N files (default: 30)")
    sp_scan.set_defaults(func=cmd_scan)

    sp_cleanup = sub.add_parser("cleanup", help="Safe cleanup (temp/caches). Default: .")
    sp_cleanup.add_argument("--yes", action="store_true", help="Actually delete (default: False)")
    sp_cleanup.add_argument("--browser-cache", action="store_true", help="Also cleanup browser cache (default: False)")
    sp_cleanup.add_argument("--nvidia-app-cache", action="store_true", help="Also cleanup Nvidia app cache (default: False)")
    sp_cleanup.set_defaults(func=cmd_cleanup)

    sp_backup = sub.add_parser("backup-browsers", help="Backup browser profiles to disk")
    sp_backup.add_argument("--dest", default=r"D:\Backups\Browsers", help="Destination directory (default: D:\Backups\Browsers)")
    sp_backup.set_defaults(func=cmd_backup)
```



```

sp_wu = sub.add_parser("winupdate-cache", help="Reset Windows Update download
sp_wu.add_argument("--yes", action="store_true", help="Execute (otherwise dry
sp_wu.set_defaults(func=cmd_winupdate_cache)

sp_trim = sub.add_parser("trim", help="Run SSD TRIM (Optimize-Volume) for a d
sp_trim.add_argument("--drive", default="C", help="Drive letter (default C).")
sp_trim.set_defaults(func=cmd_trim)

args = ap.parse_args()
args.func(args)

if __name__ == "__main__":
    main()
...

collector_py = r'''# win_collect_session.py
# Collect local logs + scripts + outputs into a single folder/zip.
# It does NOT read any ChatGPT history (that's only available via ChatGPT export)
#
# Usage:
#   python win_collect_session.py --dest D:\Backups\WinSession
#   python win_collect_session.py --dest D:\Backups\WinSession --zip
#
# What it grabs:
# - PowerShell history (PSReadLine ConsoleHost_history.txt)
# - Windows version / hardware summary (basic)
# - Recent readiness reports on Desktop (win11_readiness*.json/txt)
# - This tool files (if run from the pack folder)
#
# Optional: you can add your own files into the created session folder before zip

from __future__ import annotations

import argparse
import json
import os
import shutil
import subprocess
import sys
from datetime import datetime
from pathlib import Path
from typing import List, Tuple

```

```
def decode_best_effort(b: bytes) -> str:
    if b is None:
        return ""
    for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
        try:
            return b.decode(enc)
        except Exception:
            continue
    return b.decode("utf-8", errors="replace")

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
    p = subprocess.run(cmd, capture_output=True)
    out = decode_best_effort(p.stdout).strip()
    err = decode_best_effort(p.stderr).strip()
    return p.returncode, out, err

def copy_if_exists(src: Path, dst: Path):
    if not src.exists():
        return False
    dst.parent.mkdir(parents=True, exist_ok=True)
    if src.is_dir():
        shutil.copytree(src, dst, dirs_exist_ok=True)
    else:
        shutil.copy2(src, dst)
    return True

def find_ps_history_candidates() -> List[Path]:
    appdata = Path(os.environ.get("APPDATA", ""))
    return [
        appdata / "Microsoft" / "Windows" / "PowerShell" / "PSReadLine" / "ConsoleHost_history.txt",
        appdata / "Microsoft" / "PowerShell" / "PSReadLine" / "ConsoleHost_history.txt"
    ]

def export_installed_programs(out_file: Path):
    ps = r'''
$ErrorActionPreference = "SilentlyContinue"
$k1 = "HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\*"
'''
```

```

$k2 = "HKLM:\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*"
$k3 = "HKCU:\Software\Microsoft\Windows\CurrentVersion\Uninstall\*"
$app = @()
foreach ($k in @($k1,$k2,$k3)) {
    Get-ItemProperty $k | ForEach-Object {
        if ($_.DisplayName) {
            $app += [pscustomobject]@{
                Name = $_.DisplayName
                Version = $_.DisplayVersion
                Publisher = $_.Publisher
                InstallLocation = $_.InstallLocation
                UninstallString = $_.UninstallString
            }
        }
    }
}
$app | Sort-Object Name -Unique | ConvertTo-Json -Depth 4
...

code, out, err = run_cmd(["powershell", "-NoProfile", "-ExecutionPolicy", "By
out_file.write_text(out if out else json.dumps({"error": err, "code": code},

def export_system_summary(out_file: Path):
    code, out, err = run_cmd(["cmd", "/c", "systeminfo"])
    out_file.write_text(out if out else f"error: {err}", encoding="utf-8")

def main():
    if os.name != "nt":
        print("This tool is for Windows.")
        sys.exit(1)

    ap = argparse.ArgumentParser("win_collect_session")
    ap.add_argument("--dest", default=r"D:\Backups\WinSession", help="Where to pl
    ap.add_argument("--zip", action="store_true", help="Also zip the session fold
    args = ap.parse_args()

    stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    dest_root = Path(args.dest).expanduser().resolve()
    session_dir = dest_root / f"win_session_{stamp}"
    session_dir.mkdir(parents=True, exist_ok=True)

    # 1) PowerShell history

```

```
hist_dir = session_dir / "powershell_history"
hist_dir.mkdir(exist_ok=True)
copied = []
for cand in find_ps_history_candidates():
    dst = hist_dir / cand.name
    if copy_if_exists(cand, dst):
        copied.append(str(cand))
(hist_dir / "sources.txt").write_text("\n".join(copied) if copied else "No PS

# 2) System summary
export_system_summary(session_dir / "systeminfo.txt")

# 3) Installed programs
export_installed_programs(session_dir / "installed_programs.json")

# 4) Grab readiness reports from Desktop
desktop = Path(os.environ.get("USERPROFILE", "")) / "Desktop"
rep_dir = session_dir / "reports"
rep_dir.mkdir(exist_ok=True)
if desktop.exists():
    for pat in ("win11_readiness*.txt", "win11_readiness*.json", "win11_readi
        for f in desktop.glob(pat):
            copy_if_exists(f, rep_dir / f.name)

# 5) Copy current folder (pack scripts) into session if present
cur = Path.cwd()
pack_dir = session_dir / "pack_files"
for name in ("win_maintain.py", "win_collect_session.py", "README.md", "CHECK
    src = cur / name
    if src.exists():
        copy_if_exists(src, pack_dir / name)

# Manifest
manifest = {
    "created_at": datetime.now().isoformat(timespec="seconds"),
    "session_dir": str(session_dir),
    "notes": [
        "ChatGPT chat history is NOT collected by this script.",
        "If you need ChatGPT logs: export from ChatGPT UI and copy into this
    ],
}
(session_dir / "manifest.json").write_text(json.dumps(manifest, ensure_ascii=
```

```

zip_path = None
if args.zip:
    zip_path = str(dest_root / f"win_session_{stamp}.zip")
    shutil.make_archive(zip_path[:-4], "zip", root_dir=str(session_dir))

print(f"Session saved to: {session_dir}")
if zip_path:
    print(f"Zipped to: {zip_path}")

if __name__ == "__main__":
    main()
...

checklist_md = r'''# Чек-лист бэкапа перед “рискованными” действиями (апгрейд / ч

## 0) Главное правило
Если есть только один диск с данными – это не бэкап, это “надежда”.
Нормальный минимум: копия на D:/внешний диск/облако + проверка что открывается

---

## 1) Документы/проекты (обязательное)
Скопируй на D:\Backups\ или внешний диск:
- `C:\Users\<you>\Desktop`
- `Documents` / `Downloads` (если там рабочие файлы)
- Папки проектов: `My-Projects`, `IdeaProjects`, `way_of_warrior_`, `Foocus`, e
- Конфиги/ключи: `.ssh`, `.gitconfig` (если используешь)
- Парсеры/скрипты (все `.py`, `.bat`, `.ps1`)

Проверка: открой пару файлов на месте бэкапа.

---

## 2) Браузеры: закладки/пароли/сессии
### Самый безопасный путь
1) Включить синхронизацию аккаунта:
- Edge – Microsoft account
- Yandex – Yandex account
- Opera – Opera Sync

2) Экспортировать закладки (HTML) в отдельную папку бэкапа.

```

Если хочешь “как есть” (со сессиями/куками)

Сделай копию профиля браузера (это уже чувствительные данные):

- Edge: `%LOCALAPPDATA%\Microsoft\Edge\User Data`
- Yandex: `%LOCALAPPDATA%\Yandex\YandexBrowser\User Data`
- Opera: `%APPDATA%\Opera Software\Opera Stable`

⚠ В этих папках могут быть:

- cookies, сессии, токены (то есть “входы”)
- сохранённые пароли (в связке с Windows DPAPI)

****Рекомендация:**** хранить бэкап на диске, который ты контролируешь (не в публично

3) Мессенджеры/клиенты

- Telegram Desktop: `Settings → Advanced → Export Telegram data` (по желанию)
- Discord/Steam – обычно достаточно логинов/2FA, но если есть важные файлы/скрины

4) Драйверы и список программ

- Список программ (на всякий): экспорт через `win_collect_session.py` (installed_
- Если драйвер редкий – скачай заранее сетевой драйвер/чипсет для своей материнки

5) Что точно НЕ делать “на удачу”

- Не удалять руками `%LOCALAPPDATA%\Packages` целиком
- Не удалять `C:\Windows\Installer` / `WinSxS` руками
- Не удалять `Program Files` “папками”, если не уверен

6) Мини-план восстановления

- Флешка Windows 10/11 (Media Creation Tool)
- Доступ к BIOS/UEFI (знать кнопку входа)
- Пароли/2FA к основным аккаунтам (почта, облако)

7) Для ноутбука (Win11)

То же самое, но плюсом:

- BitLocker: сохранить Recovery Key (если включён)

- Проверить свободное место (под апгрейд надо 25–30 ГБ)
- ...

prompt_txt = r'''Сделай из этого одну “сборку” для обслуживания Windows (PC + ноу

Контекст:

- У меня Windows 10 Home (build 19045), UEFI, GPT на системном диске.
- Идём по пути: оставить Win10 и оптимизировать/почистить/ускорить.
- Параллельно готовим “план Б”: установка Win11 в обход требований (без TPM/secur

Что нужно от тебя:

- 1) Сформировать пошаговый чек-лист оптимизации Win10 (безопасно, без поломки сист
- 2) Сформировать чек-лист полного бэкапа (документы/проекты/браузеры/ключи/конфиги
- 3) Объединить логику скриптов:
 - анализ, что жрёт место (Local/Roaming, ProgramData, Win update cache),
 - безопасная очистка (Temp, CrashDumps, SquirrelTemp, WinUpdate cache),
 - чистка кэшей браузеров без удаления профиля,
 - опционально: чистка NVIDIA app UpdateFramework.
- 4) Сделать “one-click” запуск через .bat:
 - scan -> отчёт,
 - cleanup (dry-run),
 - cleanup (execute),
 - backup browsers,
 - collect session logs to zip.

Ограничения:

- По умолчанию ничего не удалять (dry-run).
- Удаление только по флагу подтверждения.
- Никаких удалений системных папок Windows вручную (WinSxS/Installer и т.п.)
- ...

readme = r'''# win_maintain_pack_v2

Единая “сборка” под Windows 10/11:

- Анализ диска (“кто жрёт место”)
- Безопасная очистка (dry-run по умолчанию)
- Очистка кэшей браузеров (не трогая профили)
- Опционально: NVIDIA app UpdateFramework cache
- Сбор “сессии” в один архив (PowerShell history + systeminfo + installed program


Быстрые команды

Открой PowerShell в папке со скриптами:

```
### 1) Анализ  
```powershell  
python .\win_maintain.py scan --outdir .
```


## 2) Очистка (сначала dry-run)

```
powershell
python .\win_maintain.py cleanup --outdir .
```

Всегда показывать подробности |  Копировать код


## 3) Очистка (выполнить)

```
powershell
python .\win_maintain.py cleanup --yes --outdir .
```

Всегда показывать подробности |  Копировать код

## 4) Кэши браузеров (закрыть браузеры!)


```
powershell
python .\win_maintain.py cleanup --yes --browser-cache --outdir .
```

Всегда показывать подробности |  Копировать код

## 5) NVIDIA app UpdateFramework (если не пользуешься NVIDIA app)


Запускать лучше от администратора:

```
powershell
python .\win_maintain.py cleanup --yes --nvidia-app-cache --outdir .
```

Всегда показывать подробности |  Копировать код


## 6) Бэкап профилей браузеров

```
powershell
python .\win_maintain.py backup-browsers --dest D:\Backups\Browsers
```

Всегда показывать подробности |  Копировать код

## 7) Собрать “сессию” в одну папку/zip

```
powershell
python .\win_collect_session.py --dest D:\Backups\WinSession --zip
```

Всегда показывать подробности |  Копировать код

## Важно



- ChatGPT-историю этот сборщик НЕ достанет (это делается экспортом в интерфейсе ChatGPT).
  - Бэкапы браузеров содержат чувствительные данные (куки/сессии). Храни их в безопасном месте.
- ...

## Batch files

```
bat_scan = r'''@echo off
setlocal
cd /d "%~dp0"
python win_maintain.py scan --outdir .
pause
'''
```

```
bat_cleanup_dry = r'''@echo off
setlocal
cd /d "%~dp0"
python win_maintain.py cleanup --outdir .
pause
'''
```

```
bat_cleanup_yes = r'''@echo off
setlocal
cd /d "%~dp0"
python win_maintain.py cleanup --yes --outdir .
pause
'''
```

```
bat_cleanup_browsers = r'''@echo off
setlocal
cd /d "%~dp0"
echo Close Edge / Yandex / Opera before continuing.
pause
python win_maintain.py cleanup --yes --browser-cache --outdir .
pause
'''
```

```
bat_cleanup_nvidia = r'''@echo off
setlocal
cd /d "%~dp0"
echo Close NVIDIA app before continuing. Recommended to run this .bat as
```

Administrator.

pause

python win\_maintain.py cleanup --yes --nvidia-app-cache --outdir .

pause

'''

bat\_backup\_browsers = r'''@echo off

setlocal

cd /d "%~dp0"

python win\_maintain.py backup-browsers --dest D:\Backups\Browsers

pause

'''

bat\_collect\_zip = r'''@echo off

setlocal

cd /d "%~dp0"

python win\_collect\_session.py --dest D:\Backups\WinSession --zip

pause

'''

## Write files

files = {

"win\_maintain.py": win\_maintain\_v2,

"win\_collect\_session.py": collector\_py,

"CHECKLIST\_backup.md": checklist\_md,

"PROMPT\_FOR\_NEW\_CHAT.txt": prompt\_txt,

"README.md": readme,

"01\_scan.bat": bat\_scan,

"02\_cleanup\_dryrun.bat": bat\_cleanup\_dry,

"03\_cleanup\_execute.bat": bat\_cleanup\_yes,

"04\_cleanup\_browser\_caches.bat": bat\_cleanup\_browsers,

"05\_cleanup\_nvidia\_app\_cache.bat": bat\_cleanup\_nvidia,

"06\_backup\_browsers.bat": bat\_backup\_browsers,

"07\_collect\_session\_zip.bat": bat\_collect\_zip,

}

for fn, content in files.items():

with open(os.path.join(base\_dir, fn), "w", encoding="utf-8") as f:

f.write(content)

```
zip_path = "/mnt/data/win_maintain_pack_v2.zip"
with zipfile.ZipFile(zip_path, "w", compression=zipfile.ZIP_DEFLATED) as z:
 for fn in files.keys():
 z.write(os.path.join(base_dir, fn), arcname=fn)
```

zip\_path

Всегда показывать подробности

 Копировать код

Проанализировано ^

python

```
import os, zipfile, shutil
from datetime import datetime
```

Всегда показывать подробности

 Копировать код

```
base_dir = "/mnt/data/win_maintain_pack_v2"
os.makedirs(base_dir, exist_ok=True)
```

```
win_maintain_v2 = """# win_maintain.py (v2)
Windows 10/11 maintenance helper: scan disk hogs, backup browser profiles, and
No external deps. Python 3.9+ recommended.
#
Safety:
- Default cleanup mode is DRY-RUN (shows what would be deleted).
- Only deletes known cache/temp locations unless you explicitly enable extra fl
- Skips symlinks/reparse points while scanning to avoid loops.
#
Usage examples:
python win_maintain.py scan
python win_maintain.py cleanup (dry-run)
python win_maintain.py cleanup --yes (execute)
python win_maintain.py cleanup --yes --browser-cache
python win_maintain.py cleanup --yes --nvidia-app-cache
python win_maintain.py backup-browsers --dest D:\\Backups\\Browsers
python win_maintain.py winupdate-cache --yes
#
Tip: run PowerShell as Administrator for best results.
```

```
from __future__ import annotations
```

```
import argparse
```

```
import csv
import ctypes
import heapq
import json
import os
import shutil
import subprocess
import sys
import time

from dataclasses import dataclass, asdict
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Tuple

REPARSE_POINT_ATTR = 0x0400 # FILE_ATTRIBUTE_REPARSE_POINT

def is_admin() -> bool:
 try:
 return bool(ctypes.windll.shell32.IsUserAnAdmin())
 except Exception:
 return False

def decode_best_effort(b: bytes) -> str:
 if b is None:
 return ""
 for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
 try:
 return b.decode(enc)
 except Exception:
 continue
 return b.decode("utf-8", errors="replace")

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
 p = subprocess.run(cmd, capture_output=True)
 out = decode_best_effort(p.stdout).strip()
 err = decode_best_effort(p.stderr).strip()
 return p.returncode, out, err
```

```
def run_powershell(ps: str) -> Tuple[int, str, str]:
 prefix = "[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new(); $Pro
 return run_cmd(["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-C

def format_gb(num_bytes: float) -> str:
 return f"{num_bytes / (1024**3):.2f} GB"

def is_reparse_point(path: str) -> bool:
 try:
 st = os.stat(path, follow_symlinks=False)
 attrs = getattr(st, "st_file_attributes", 0)
 return bool(attrs & REPARSE_POINT_ATTR)
 except Exception:
 return False

def list_drives() -> List[str]:
 drives = []
 for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
 p = f"{c}:\\\\"
 if os.path.exists(p):
 drives.append(p)
 return drives

@dataclass
class ScanStats:
 root: str
 depth: int
 files_scanned: int = 0
 dirs_scanned: int = 0
 bytes_total: int = 0
 errors: int = 0
 denied: int = 0
 skipped_reparse: int = 0
 elapsed_sec: float = 0.0

@dataclass
class ScanResult:
 stats: ScanStats
```

```
top_dirs_level1: List[Tuple[str, int]]
top_dirs_level2: List[Tuple[str, int]]
top_files: List[Tuple[str, int]]

def scan_root(root: str, depth: int, top_dirs: int, top_files: int) -> ScanResult
 t0 = time.time()
 root = os.path.abspath(root)
 stats = ScanStats(root=root, depth=depth)

 lvl1: Dict[str, int] = {}
 lvl2: Dict[str, int] = {}
 file_heap: List[Tuple[int, str]] = []

 stack = [root]
 root_parts_len = len(Path(root).parts)

 def add_file(sz: int, p: str):
 if top_files <= 0:
 return
 if len(file_heap) < top_files:
 heapq.heappush(file_heap, (sz, p))
 else:
 if sz > file_heap[0][0]:
 heapq.heapreplace(file_heap, (sz, p))

 while stack:
 cur = stack.pop()
 stats.dirs_scanned += 1
 try:
 with os.scandir(cur) as it:
 for entry in it:
 try:
 if entry.is_symlink():
 continue
 full = entry.path
 if is_reparse_point(full):
 stats.skipped_reparse += 1
 continue

 if entry.is_dir(follow_symlinks=False):
 rel_parts = len(Path(full).parts) - root_parts_len
 if depth < 0 or rel_parts <= depth:
```

```

 stack.append(full)
 elif entry.is_file(follow_symlinks=False):
 st = entry.stat(follow_symlinks=False)
 sz = int(getattr(st, "st_size", 0))
 stats.files_scanned += 1
 stats.bytes_total += sz

 try:
 rel = os.path.relpath(full, root)
 except Exception:
 rel = full
 parts = rel.split(os.sep)
 if len(parts) >= 1:
 lvl1[parts[0]] = lvl1.get(parts[0], 0) + sz
 if len(parts) >= 2:
 k2 = os.path.join(parts[0], parts[1])
 lvl2[k2] = lvl2.get(k2, 0) + sz

 add_file(sz, full)
except PermissionError:
 stats.denied += 1
except FileNotFoundError:
 stats.errors += 1
except OSError:
 stats.errors += 1
except PermissionError:
 stats.denied += 1
except FileNotFoundError:
 stats.errors += 1
except OSError:
 stats.errors += 1

stats.elapsed_sec = round(time.time() - t0, 2)
top1 = sorted(lvl1.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top2 = sorted(lvl2.items(), key=lambda x: x[1], reverse=True)[:top_dirs]
top_files_list = sorted([(p, sz) for (sz, p) in file_heap], key=lambda x: x[1])
return ScanResult(stats=stats, top_dirs_level1=top1, top_dirs_level2=top2, top_files_list=top_files_list)

def print_drive_table() -> List[Dict[str, str]]:
 rows = []
 print("=== Drives ===")
 for d in list_drives():

```

```

try:
 du = shutil.disk_usage(d)
 used = du.total - du.free
 print(f"{d} Total: {format_gb(du.total)} | Used: {format_gb(used)} |")
 rows.append({"drive": d, "total_gb": f"{du.total/(1024**3):.2f}", "fr
except Exception as e:
 print(f"{d} (error reading usage: {e})")
 rows.append({"drive": d, "error": str(e)})
print()
return rows

```

```
from dataclasses import dataclass
```

```
@dataclass
```

```
class CleanupAction:
```

```
 name: str
```

```
 description: str
```

```
 targets: List[str]
```

```
 needs_admin: bool = False
```

```
def expand_globs(patterns: List[str]) -> List[Path]:
```

```
 out: List[Path] = []
```

```
 for pat in patterns:
```

```
 pat2 = os.path.expandvars(pat)
```

```
 p = Path(pat2)
```

```
 if "*" in pat2 or "?" in pat2:
```

```
 parent = p.parent
```

```
 if parent.exists():
```

```
 out.extend(list(parent.glob(p.name)))
```

```
 else:
```

```
 out.append(p)
```

```
 else:
```

```
 out.append(p)
```

```
 uniq, seen = [], set()
```

```
 for p in out:
```

```
 s = str(p).lower()
```

```
 if s not in seen:
```

```
 uniq.append(p)
```

```
 seen.add(s)
```

```
 return uniq
```



```
def remove_path(p: Path) -> Tuple[bool, str]:
 try:
 if not p.exists():
 return True, "not_found"
 if p.is_symlink():
 return True, "skip_symlink"
 if is_reparse_point(str(p)):
 return True, "skip_reparse_point"
 if p.is_file():
 p.unlink()
 return True, "deleted_file"
 if p.is_dir():
 shutil.rmtree(p, ignore_errors=False)
 return True, "deleted_dir"
 return True, "unknown_type"
 except PermissionError as e:
 return False, f"permission_denied: {e}"
 except Exception as e:
 return False, f"error: {e}"

def cleanup(actions: List[CleanupAction], yes: bool, outdir: Path) -> Dict[str, o
report = {
 "generated_at": datetime.now().isoformat(timespec="seconds"),
 "is_admin": is_admin(),
 "mode": "EXECUTE" if yes else "DRY_RUN",
 "actions": [],
}

for act in actions:
 act_entry = {"name": act.name, "description": act.description, "needs_admin":
 if act.needs_admin and not is_admin():
 act_entry["skipped"] = "needs_admin"
 report["actions"].append(act_entry)
 continue

 paths = expand_globs(act.targets)
 for p in paths:
 item = {"path": str(p), "exists": p.exists()}
 if yes:
 ok, msg = remove_path(p)
 item["result"] = msg
```

```

 item["ok"] = ok
 act_entry["items"].append(item)

 report["actions"].append(act_entry)

if yes:
 code, out, err = run_powershell("try { Clear-RecycleBin -Force -ErrorActi
 report["recycle_bin"] = {"code": code, "out": out, "err": err}

outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
(outdir / f"cleanup_report_{ts}.json").write_text(json.dumps(report, ensure_a
return report

def winupdate_cache_reset(yes: bool, outdir: Path) -> Dict[str, object]:
 report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "mode
 report["target"] = os.path.expandvars(r"%windir%\SoftwareDistribution\Downl

if not yes:
 report["note"] = "Would stop wuauserv/bits, delete Download cache, then s
 (outdir / f"winupdate_cache_dryrun_{datetime.now().strftime('%Y%m%d_%H%M%
 json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
 return report

if not is_admin():
 report["error"] = "Run as Administrator for this action."
 (outdir / f"winupdate_cache_error_{datetime.now().strftime('%Y%m%d_%H%M%S
 json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
 return report

cmds = [
 ["net", "stop", "wuauserv"],
 ["net", "stop", "bits"],
 ["powershell", "-NoProfile", "-ExecutionPolicy", "Bypass", "-Command",
 r"Remove-Item -Recurse -Force \"$env:windir\SoftwareDistribution\Downl
 ["net", "start", "bits"],
 ["net", "start", "wuauserv"],
]
results = []
for c in cmds:

```

```
code, out, err = run_cmd(c)
results.append({"cmd": " ".join(c), "code": code, "out": out, "err": err})
report["results"] = results

(outdir / f"winupdate_cache_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json")
 json.dumps(report, ensure_ascii=False, indent=2), encoding="utf-8"
)
return report

def backup_browsers(dest: str) -> Dict[str, object]:
 local = Path(os.environ.get("LOCALAPPDATA", ""))
 roam = Path(os.environ.get("APPDATA", ""))
 dest_root = Path(dest).expanduser().resolve()
 stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
 out = dest_root / f"browsers_backup_{stamp}"
 out.mkdir(parents=True, exist_ok=True)

def copytree_fast(src: Path, dst: Path):
 ignore = shutil.ignore_patterns(
 "Cache", "Code Cache", "GPUCache", "ShaderCache",
 "Media Cache", "GrShaderCache", "DawnCache",
 "Service Worker\\\\\\CacheStorage", "Service Worker\\\\\\ScriptCache",
 "Crashpad", "Crash Reports"
)
 shutil.copytree(src, dst, dirs_exist_ok=True, ignore=ignore)

sources = {
 "Edge_UserData": local / "Microsoft" / "Edge" / "User Data",
 "Yandex_UserData": local / "Yandex" / "YandexBrowser" / "User Data",
 "Opera_Stable_Roaming": roam / "Opera Software" / "Opera Stable",
 "OperaGX_Stable_Roaming": roam / "Opera Software" / "Opera GX Stable",
}

report = {"saved_to": str(out), "items": []}
for name, src in sources.items():
 if src.exists():
 dst = out / name
 try:
 copytree_fast(src, dst)
 report["items"].append({"name": name, "src": str(src), "dst": str
 except Exception as e:
 report["items"].append({"name": name, "src": str(src), "dst": str
```

```

 else:
 report["items"].append({"name": name, "src": str(src), "status": "not

(out / "manifest.json").write_text(json.dumps(report, ensure_ascii=False, ind
return report

def optimize_ssd(drive_letter: str) -> Dict[str, object]:
 report = {"generated_at": datetime.now().isoformat(timespec="seconds"), "driv
 if not is_admin():
 report["error"] = "Run as Administrator for TRIM."
 return report
 ps = f"try {{ Optimize-Volume -DriveLetter {drive_letter} -ReTrim -Verbose }}"
 code, out, err = run_powershell(ps)
 report.update({"code": code, "out": out, "err": err})
 return report

def cmd_scan(args: argparse.Namespace):
 drives_info = print_drive_table()

 roots = args.roots or []
 if not roots:
 user = os.environ.get("USERPROFILE", "")
 roots = [
 os.path.join(user, "AppData", "Local"),
 os.path.join(user, "AppData", "Roaming"),
 os.path.join(os.environ.get("windir", r"C:\Windows"), "SoftwareDistr
 os.path.join(os.environ.get("windir", r"C:\Windows"), "Temp"),
]

 results: List[ScanResult] = []
 for r in roots:
 if not r:
 continue
 r = os.path.expandvars(r)
 if not os.path.exists(r):
 print(f"[skip] Not found: {r}")
 continue
 print(f"=== Scanning: {r} (depth={args.depth}) ===")
 res = scan_root(r, depth=args.depth, top_dirs=args.top, top_files=args.fi
 results.append(res)

```

```

st = res.stats
print(f"Scanned dirs={st.dirs_scanned}, files={st.files_scanned}, total={
 f"denied={st.denied}, errors={st.errors}, skipped_reparse={st.skipp

print("--- Top folders (level 1) ---")
for name, sz in res.top_dirs_level1:
 print(f"{name:<45} {format_gb(sz)}")
print("\n--- Top folders (level 2) ---")
for name, sz in res.top_dirs_level2:
 print(f"{name:<45} {format_gb(sz)}")
print("\n--- Top files ---")
for p, sz in res.top_files:
 print(f"{format_gb(sz):>10} {p}")
print("\n" + "=" * 70 + "\n")

outdir = Path(args.outdir).resolve()
outdir.mkdir(parents=True, exist_ok=True)
ts = datetime.now().strftime("%Y%m%d_%H%M%S")
base = outdir / f"scan_report_{ts}"

payload = {
 "generated_at": datetime.now().isoformat(timespec="seconds"),
 "is_admin": is_admin(),
 "drives": drives_info,
 "results": [
 {"stats": asdict(r.stats), "top_dirs_level1": r.top_dirs_level1, "top
 for r in results
],
}
(base.with_suffix(".json")).write_text(json.dumps(payload, ensure_ascii=False

with open(base.with_suffix("_topdirs.csv"), "w", encoding="utf-8-sig", newlin
w = csv.writer(f, delimiter=";")
w.writerow(["root", "level", "path", "bytes", "gb"])
for r in results:
 root = r.stats.root
 for p, sz in r.top_dirs_level1:
 w.writerow([root, 1, p, sz, round(sz / (1024**3), 3)])
 for p, sz in r.top_dirs_level2:
 w.writerow([root, 2, p, sz, round(sz / (1024**3), 3)])

print(f"Saved:\n- {base.with_suffix('.json')}\n- {base.with_suffix('_topdir

```

```
def build_cleanup_actions(include_browser_cache: bool, include_nvidia_app: bool):
 local = r"%LOCALAPPDATA%"
 temp = r"%TEMP%"

 actions = [
 CleanupAction(
 name="Temp (user)",
 description="Clear user TEMP files (safe).",
 targets=[temp + r"*"],
 needs_admin=False,
),
 CleanupAction(
 name="CrashDumps",
 description="Clear Windows Error Reporting crash dumps for user (safe)",
 targets=[local + r"\CrashDumps*"],
 needs_admin=False,
),
 CleanupAction(
 name="SquirrelTemp",
 description="Clear installer temp used by Squirrel-based updaters (safe)",
 targets=[local + r"\SquirrelTemp*"],
 needs_admin=False,
),
 CleanupAction(
 name="NVIDIA cache (ProgramData)",
 description="Clear NVIDIA downloader + NV_Cache (safe).",
 targets=[r"C:\ProgramData\NVIDIA Corporation\Downloader*", r"C:\ProgramData\NVIDIA Corporation\NV_Cache*"],
 needs_admin=True,
),
]

 if include_nvidia_app:
 actions.append(
 CleanupAction(
 name="NVIDIA app UpdateFramework",
 description="Clear NVIDIA app UpdateFramework cache (will re-download)",
 targets=[
 r"C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework*",
 r"C:\ProgramData\NVIDIA Corporation\NVIDIA app\Installer*",
],
 needs_admin=True,
)
)
```

```

)

if include_browser_cache:
 actions.extend([
 CleanupAction(
 name="Yandex Browser cache",
 description="Clear Yandex browser caches only (close browser first)",
 targets=[
 local + r"\\Yandex\\YandexBrowser\\User Data*\\Cache*",
 local + r"\\Yandex\\YandexBrowser\\User Data*\\Code Cache\\",
 local + r"\\Yandex\\YandexBrowser\\User Data*\\GPUCache*",
 local + r"\\Yandex\\YandexBrowser\\User Data*\\Service Work
 local + r"\\Yandex\\YandexBrowser\\User Data*\\Service Work
],
 needs_admin=False,
),
 CleanupAction(
 name="Opera cache",
 description="Clear Opera caches only (close browser first).",
 targets=[
 local + r"\\Opera Software\\Opera Stable\\Cache*",
 local + r"\\Opera Software\\Opera Stable\\Code Cache*",
 local + r"\\Opera Software\\Opera Stable\\GPUCache*",
 local + r"\\Opera Software\\Opera GX Stable\\Cache*",
 local + r"\\Opera Software\\Opera GX Stable\\Code Cache*",
 local + r"\\Opera Software\\Opera GX Stable\\GPUCache*",
],
 needs_admin=False,
),
 CleanupAction(
 name="Edge cache",
 description="Clear Edge caches only (close browser first).",
 targets=[
 local + r"\\Microsoft\\Edge\\User Data*\\Cache*",
 local + r"\\Microsoft\\Edge\\User Data*\\Code Cache*",
 local + r"\\Microsoft\\Edge\\User Data*\\GPUCache*",
 local + r"\\Microsoft\\Edge\\User Data*\\Service Worker\\Ca
 local + r"\\Microsoft\\Edge\\User Data*\\Service Worker\\Sc
],
 needs_admin=False,
),
])

```

```
 return actions

def cmd_cleanup(args: argparse.Namespace):
 outdir = Path(args.outdir).resolve()
 actions = build_cleanup_actions(
 include_browser_cache=args.browser_cache,
 include_nvidia_app=args.nvidia_app_cache
)

 print(f"Mode: {'EXECUTE' if args.yes else 'DRY_RUN'}")
 if args.browser_cache:
 print("Including browser caches: YES (close browsers before executing)")
 if args.nvidia_app_cache:
 print("Including NVIDIA app UpdateFramework cache: YES (admin + NVIDIA ap

 rep = cleanup(actions, yes=args.yes, outdir=outdir)
 print(f"Report saved into: {outdir}")
 if rep.get("mode") == "DRY_RUN":
 print("Dry-run complete. Re-run with --yes to actually delete.")
 else:
 print("Cleanup executed.")

def cmd_backup(args: argparse.Namespace):
 rep = backup_browsers(args.dest)
 print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_winupdate_cache(args: argparse.Namespace):
 outdir = Path(args.outdir).resolve()
 rep = winupdate_cache_reset(yes=args.yes, outdir=outdir)
 print(json.dumps(rep, ensure_ascii=False, indent=2))

def cmd_trim(args: argparse.Namespace):
 rep = optimize_ssd(args.drive)
 print(json.dumps(rep, ensure_ascii=False, indent=2))

def main():
 if os.name != "nt":
 print("This tool is for Windows.")
```



```

sys.exit(1)

ap = argparse.ArgumentParser("win_maintain", description="Scan + safe cleanup
ap.add_argument("--outdir", default=".", help="Output dir for reports (default

sub = ap.add_subparsers(dest="cmd", required=True)

sp_scan = sub.add_parser("scan", help="Scan disk hogs under common roots (or
sp_scan.add_argument("--roots", nargs="*", default=[], help="Roots to scan (d
sp_scan.add_argument("--depth", type=int, default=6, help="Relative depth (de
sp_scan.add_argument("--top", type=int, default=25, help="Top N folders (defa
sp_scan.add_argument("--files", type=int, default=30, help="Top N files (defa
sp_scan.set_defaults(func=cmd_scan)

sp_cleanup = sub.add_parser("cleanup", help="Safe cleanup (temp/caches). Defa
sp_cleanup.add_argument("--yes", action="store_true", help="Actually delete (
sp_cleanup.add_argument("--browser-cache", action="store_true", help="Also cl
sp_cleanup.add_argument("--nvidia-app-cache", action="store_true", help="Also
sp_cleanup.set_defaults(func=cmd_cleanup)

sp_backup = sub.add_parser("backup-browsers", help="Backup browser profiles t
sp_backup.add_argument("--dest", default=r"D:\\Backups\\Browsers", help="Dest
sp_backup.set_defaults(func=cmd_backup)

sp_wu = sub.add_parser("winupdate-cache", help="Reset Windows Update download
sp_wu.add_argument("--yes", action="store_true", help="Execute (otherwise dry
sp_wu.set_defaults(func=cmd_winupdate_cache)

sp_trim = sub.add_parser("trim", help="Run SSD TRIM (Optimize-Volume) for a d
sp_trim.add_argument("--drive", default="C", help="Drive letter (default C).
sp_trim.set_defaults(func=cmd_trim)

args = ap.parse_args()
args.func(args)

if __name__ == "__main__":
 main()
"""

collector_py = """# win_collect_session.py
Collect local logs + scripts + outputs into a single folder/zip.
It does NOT read any ChatGPT history (that's only available via ChatGPT export)

```

```
#
Usage:
python win_collect_session.py --dest D:\\Backups\\WinSession
python win_collect_session.py --dest D:\\Backups\\WinSession --zip
#
What it grabs:
- PowerShell history (PSReadLine ConsoleHost_history.txt)
- Windows version / hardware summary (basic)
- Recent readiness reports on Desktop (win11_readiness*.json/txt)
- This tool files (if run from the pack folder)
#
Optional: you can add your own files into the created session folder before zip

from __future__ import annotations

import argparse
import json
import os
import shutil
import subprocess
import sys
from datetime import datetime
from pathlib import Path
from typing import List, Tuple

def decode_best_effort(b: bytes) -> str:
 if b is None:
 return ""
 for enc in ("utf-8", "utf-8-sig", "cp866", "cp1251", "mbcs"):
 try:
 return b.decode(enc)
 except Exception:
 continue
 return b.decode("utf-8", errors="replace")

def run_cmd(cmd: List[str]) -> Tuple[int, str, str]:
 p = subprocess.run(cmd, capture_output=True)
 out = decode_best_effort(p.stdout).strip()
 err = decode_best_effort(p.stderr).strip()
 return p.returncode, out, err
```

```

def copy_if_exists(src: Path, dst: Path):
 if not src.exists():
 return False
 dst.parent.mkdir(parents=True, exist_ok=True)
 if src.is_dir():
 shutil.copytree(src, dst, dirs_exist_ok=True)
 else:
 shutil.copy2(src, dst)
 return True

def find_ps_history_candidates() -> List[Path]:
 appdata = Path(os.environ.get("APPDATA", ""))
 return [
 appdata / "Microsoft" / "Windows" / "PowerShell" / "PSReadLine" / "Consol
 appdata / "Microsoft" / "PowerShell" / "PSReadLine" / "ConsoleHost_histor
]

def export_installed_programs(out_file: Path):
 ps = r'''
$ErrorActionPreference = "SilentlyContinue"
$k1 = "HKLM:\\Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall*"
$k2 = "HKLM:\\Software\\WOW6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstal
$k3 = "HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall*"
$app = @(
foreach ($k in @($k1,$k2,$k3)) {
 Get-ItemProperty $k | ForEach-Object {
 if ($_.DisplayName) {
 $app += [pscustomobject]@{
 Name = $_.DisplayName
 Version = $_.DisplayVersion
 Publisher = $_.Publisher
 InstallLocation = $_.InstallLocation
 UninstallString = $_.UninstallString
 }
 }
 }
}
)
$app | Sort-Object Name -Unique | ConvertTo-Json -Depth 4
'''

 code, out, err = run_cmd(["powershell", "-NoProfile", "-ExecutionPolicy", "By

```

```
out_file.write_text(out if out else json.dumps({"error": err, "code": code},

def export_system_summary(out_file: Path):
 code, out, err = run_cmd(["cmd", "/c", "systeminfo"])
 out_file.write_text(out if out else f"error: {err}", encoding="utf-8")

def main():
 if os.name != "nt":
 print("This tool is for Windows.")
 sys.exit(1)

 ap = argparse.ArgumentParser("win_collect_session")
 ap.add_argument("--dest", default=r"D:\\Backups\\WinSession", help="Where to
 ap.add_argument("--zip", action="store_true", help="Also zip the session fold
 args = ap.parse_args()

 stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
 dest_root = Path(args.dest).expanduser().resolve()
 session_dir = dest_root / f"win_session_{stamp}"
 session_dir.mkdir(parents=True, exist_ok=True)

 # 1) PowerShell history
 hist_dir = session_dir / "powershell_history"
 hist_dir.mkdir(exist_ok=True)
 copied = []
 for cand in find_ps_history_candidates():
 dst = hist_dir / cand.name
 if copy_if_exists(cand, dst):
 copied.append(str(cand))
 (hist_dir / "sources.txt").write_text("\n".join(copied) if copied else "No P

 # 2) System summary
 export_system_summary(session_dir / "systeminfo.txt")

 # 3) Installed programs
 export_installed_programs(session_dir / "installed_programs.json")

 # 4) Grab readiness reports from Desktop
 desktop = Path(os.environ.get("USERPROFILE", "")) / "Desktop"
 rep_dir = session_dir / "reports"
 rep_dir.mkdir(exist_ok=True)
```

```

if desktop.exists():
 for pat in ("win11_readiness*.txt", "win11_readiness*.json", "win11_readi
 for f in desktop.glob(pat):
 copy_if_exists(f, rep_dir / f.name)

5) Copy current folder (pack scripts) into session if present
cur = Path.cwd()
pack_dir = session_dir / "pack_files"
pack_dir.mkdir(exist_ok=True)
for name in ("win_maintain.py", "win_collect_session.py", "README.md", "CHECK
 src = cur / name
 if src.exists():
 copy_if_exists(src, pack_dir / name)

Manifest
manifest = {
 "created_at": datetime.now().isoformat(timespec="seconds"),
 "session_dir": str(session_dir),
 "notes": [
 "ChatGPT chat history is NOT collected by this script.",
 "If you need ChatGPT logs: export from ChatGPT UI and copy into this
],
}
(session_dir / "manifest.json").write_text(json.dumps(manifest, ensure_ascii=

zip_path = None
if args.zip:
 zip_path = str(dest_root / f"win_session_{stamp}.zip")
 shutil.make_archive(zip_path[:-4], "zip", root_dir=str(session_dir))

print(f"Session saved to: {session_dir}")
if zip_path:
 print(f"Zipped to: {zip_path}")

if __name__ == "__main__":
 main()
"""

checklist_md = """# Чек-лист бэкапа перед “рискованными” действиями (апгрейд / чи

0) Главное правило
Если есть только один диск с данными — **это не бэкап**, это “надежда”.

```

Нормальный минимум: \*\*копия на D:/внешний диск/облако + проверка что открывается\*

---

### ## 1) Документы/проекты (обязательное)

Скопируй на D:\\Backups\\ или внешний диск:

- `C:\\Users\\<you>\\Desktop`
- `Documents` / `Downloads` (если там рабочие файлы)
- Папки проектов: `My-Projects`, `IdeaProjects`, `way\_of\_warrior\_\*`, `Foocus`, e`
- Конфиги/ключи: `.ssh`, `.gitconfig` (если используешь)
- Парсеры/скрипты (все `.py`, `.bat`, `.ps1`)

**\*\*Проверка:\*\*** открой пару файлов на месте бэкапа.

---

### ## 2) Браузеры: закладки/пароли/сессии

### Самый безопасный путь

1) Включить синхронизацию аккаунта:

- Edge – Microsoft account
- Yandex – Yandex account
- Opera – Opera Sync

2) Экспортировать закладки (HTML) в отдельную папку бэкапа.

### Если хочешь “как есть” (со сессиями/куками)

Сделай копию профиля браузера (это уже чувствительные данные):

- Edge: `%LOCALAPPDATA%\Microsoft\Edge\User Data`
- Yandex: `%LOCALAPPDATA%\Yandex\YandexBrowser\User Data`
- Opera: `%APPDATA%\Opera Software\Opera Stable`



В этих папках могут быть:

- cookies, сессии, токены (то есть “входы”)
- сохранённые пароли (в связке с Windows DPAPI)

**\*\*Рекомендация:\*\*** хранить бэкап на диске, который ты контролируешь (не в публично

---

### ## 3) Мессенджеры/клиенты

- Telegram Desktop: `Settings → Advanced → Export Telegram data` (по желанию)
- Discord/Steam – обычно достаточно логинов/2FA, но если есть важные файлы/скрины

---

#### ## 4) Драйверы и список программ

- Список программ (на всякий): экспорт через `win\_collect\_session.py` (installed\_
- Если драйвер редкий – скачай заранее сетевой драйвер/чипсет для своей материнки

---

#### ## 5) Что точно НЕ делать “на удачу”

- Не удалять руками `%LOCALAPPDATA%\Packages` целиком
- Не удалять `C:\\Windows\\Installer` / `WinSxS` руками
- Не удалять `Program Files` “папками”, если не уверен

---

#### ## 6) Мини-план восстановления

- Флешка Windows 10/11 (Media Creation Tool)
- Доступ к BIOS/UEFI (знать кнопку входа)
- Пароли/2FA к основным аккаунтам (почта, облако)

---

#### ## 7) Для ноутбука (Win11)

То же самое, но плюсом:

- BitLocker: сохранить Recovery Key (если включён)
- Проверить свободное место (под апгрейд надо 25–30 ГБ)

"""

prompt\_txt = """"Сделай из этого одну “сборку” для обслуживания Windows (PC + ноут

Контекст:

- У меня Windows 10 Home (build 19045), UEFI, GPT на системном диске.
- Идём по пути: оставить Win10 и оптимизировать/почистить/ускорить.
- Параллельно готовим “план Б”: установка Win11 в обход требований (без TPM/secu

Что нужно от тебя:

- 1) Сформировать пошаговый чек-лист оптимизации Win10 (безопасно, без поломки сист
- 2) Сформировать чек-лист полного бэкапа (документы/проекты/браузеры/ключи/конфиги
- 3) Объединить логику скриптов:
  - анализ, что жрёт место (Local/Roaming, ProgramData, Win update cache),
  - безопасная очистка (Temp, CrashDumps, SquirrelTemp, WinUpdate cache),
  - чистка кэшей браузеров без удаления профиля,
  - опционально: чистка NVIDIA app UpdateFramework.

#### 4) Сделать “one-click” запуск через .bat:

- scan -> отчёт,
- cleanup (dry-run),
- cleanup (execute),
- backup browsers,
- collect session logs to zip.

#### Ограничения:

- По умолчанию ничего не удалять (dry-run).
  - Удаление только по флагу подтверждения.
  - Никаких удалений системных папок Windows вручную (WinSxS/Installer и т.п.)
- """

```
readme = """# win_maintain_pack_v2
```

#### Единая “сборка” под Windows 10/11:

- Анализ диска (“кто жрёт место”)
- Безопасная очистка (dry-run по умолчанию)
- Очистка кэшей браузеров (не трогая профили)
- Опционально: NVIDIA app UpdateFramework cache
- Сбор “сессии” в один архив (PowerShell history + systeminfo + installed program

#### ## Быстрые команды

Открой PowerShell в папке со скриптами:

#### ### 1) Анализ

```
```powershell  
python .\win_maintain.py scan --outdir .
```

2) Очистка (сначала dry-run)

powershell

Всегда показывать подробности

 Копировать код

```
python .\win_maintain.py cleanup --outdir .
```

3) Очистка (выполнить)

powershell

Всегда показывать подробности

 Копировать код

```
python .\win_maintain.py cleanup --yes --outdir .
```

4) Кэши браузеров (закрыть браузеры!)

powershell

Всегда показывать подробности

 Копировать код

```
python .\win_maintain.py cleanup --yes --browser-cache --outdir .
```

5) NVIDIA app UpdateFramework (если не пользуешься NVIDIA app)

Запускать лучше от администратора:

powershell

Всегда показывать подробности

 Копировать код

```
python .\win_maintain.py cleanup --yes --nvidia-app-cache --outdir .
```

6) Бэкап профилей браузеров

powershell

Всегда показывать подробности

 Копировать код

```
python .\win_maintain.py backup-browsers --dest D:\\Backups\\Browsers
```

7) Собрать "сессию" в одну папку/zip

powershell

Всегда показывать подробности

 Копировать код

```
python .\win_collect_session.py --dest D:\\Backups\\WinSession --zip
```

Важно

- ChatGPT-историю этот сборщик НЕ достанет (это делается экспортом в интерфейсе ChatGPT).
- Бэкапы браузеров содержат чувствительные данные (куки/сессии). Храни их в безопасном месте.

```
"""
```

```
bat_scan = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
```

```
python win_maintain.py scan --outdir .
```

```
pause
```

```
"""
```

```
bat_cleanup_dry = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
```

```
python win_maintain.py cleanup --outdir .
```

```
pause
```

```
"""
```

```
bat_cleanup_yes = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
```

```
python win_maintain.py cleanup --yes --outdir .
```

```
pause
```

```
"""
```

```
bat_cleanup_browsers = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
```

```
echo Close Edge / Yandex / Opera before continuing.
```

```
pause
```

```
python win_maintain.py cleanup --yes --browser-cache --outdir .
```

```
pause
```

```
"""
```

```
bat_cleanup_nvidia = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
```

```
echo Close NVIDIA app before continuing. Recommended to run this .bat as  
Administrator.
```

```
pause
```

```
python win_maintain.py cleanup --yes --nvidia-app-cache --outdir .
```

```
pause
```

```
"""
```

```
bat_backup_browsers = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
```

```
python win_maintain.py backup-browsers --dest D:\Backups\Browsers
```

```
pause
```

```
"""
```

```
bat_collect_zip = """"@echo off
```

```
setlocal
```

```
cd /d "%~dp0"
python win_collect_session.py --dest D:\Backups\WinSession --zip
pause
"""

files = {
    "win_maintain.py": win_maintain_v2,
    "win_collect_session.py": collector_py,
    "CHECKLIST_backup.md": checklist_md,
    "PROMPT_FOR_NEW_CHAT.txt": prompt_txt,
    "README.md": readme,
    "01_scan.bat": bat_scan,
    "02_cleanup_dryrun.bat": bat_cleanup_dry,
    "03_cleanup_execute.bat": bat_cleanup_yes,
    "04_cleanup_browser_caches.bat": bat_cleanup_browsers,
    "05_cleanup_nvidia_app_cache.bat": bat_cleanup_nvidia,
    "06_backup_browsers.bat": bat_backup_browsers,
    "07_collect_session_zip.bat": bat_collect_zip,
}

for fn, content in files.items():
    with open(os.path.join(base_dir, fn), "w", encoding="utf-8") as f:
        f.write(content)

zip_path = "/mnt/data/win_maintain_pack_v2.zip"
with zipfile.ZipFile(zip_path, "w", compression=zipfile.ZIP_DEFLATED) as z:
    for fn in files.keys():
        z.write(os.path.join(base_dir, fn), arcname=fn)

zip_path
```

[Всегда показывать подробности](#)[Копировать код](#)

Результат

```
'/mnt/data/win_maintain_pack_v2.zip'
```

Тот кусок на скрине (`write files ... zipfile.ZipFile(...)`) — это **техническая “кухня”**, как я у себя собрал файлы в архив: создал `win_maintain.py` , `README.md` , а

потом упаковал их в `win_maintain_pack.zip`. Для тебя это не команда, которую надо запускать — просто “как был сделан архив”.

Почему NVIDIA “UpdateFramework” жрёт 3.56 GB и можно ли чистить

Да, это типичные ОТА-артефакты обновлений NVIDIA app (кэш скачанных пакетов/патчей/постобработки). Если ты “не особо пользуешься” NVIDIA app — можно:

Вариант А (самый чистый)

Удалить NVIDIA app через “Приложения и возможности”, драйвер оставить. Кэш уйдёт, и больше не будет накапливаться.

Вариант В (оставить, но чистить кэш)

Закрой NVIDIA app → (желательно PowerShell от админа) и чистишь папку UpdateFramework.

Ты уже видел, что там `ota-artifacts\...` — это оно.

Команда (пример):

```
powershell
```

```
Remove-Item -Recurse -Force "C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateF
```

◀   Копировать код

Почему у тебя в выводе не показалась колонка GB

Ты делал:

```
powershell
```

```
Select-Object -First 30 FullName, @{n="GB";e={...}}
```

 Копировать код

А консоль вывела только `FullName`. Частая причина — форматирование/ширина/какая-то “каша” с выводом. Попробуй так (железобетонно):

```
powershell
```

```
Get-ChildItem "C:\ProgramData\NVIDIA Corporation\NVIDIA app\UpdateFramework" -Rec  
Sort-Object Length -Descending |
```

```
Select-Object -First 30 @{n="GB";e={[math]::Round($_.Length/1GB,2)}}, FullName  
Format-Table -AutoSize
```

 Копировать код

“Запиши всю историю/скрипты/результаты в лог/архив”

Честно: историю чатов ChatGPT я из Windows не вытащу — это делается только через экспорт в интерфейсе ChatGPT (Settings → Data controls → Export data).

Но я сделал сборку, которая:

- собирает **PowerShell history** (PSReadLine),
- сохраняет systeminfo ,
- выгружает список установленных программ,
- подтягивает readiness-отчёты с Desktop,
- кладёт внутрь сами скрипты/батники.

Готовая “сборка” (v2)

Я упаковал всё в один архив: скрипты + батники + чек-лист бэкапа + “промпт для нового чата”.

Скачать: [win_maintain_pack_v2.zip](#)

Что внутри

- win_maintain.py — scan/cleanup/backup-browsers (dry-run по умолчанию)
- win_collect_session.py — собрать “сессию” в папку/zip (history+systeminfo+apps+reports)
- батники 01_...bat ... 07_...bat — one-click запуски
- CHECKLIST_backup.md — нормальный чек-лист бэкапа перед Win11 bypass/переустановкой
- PROMPT_FOR_NEW_CHAT.txt — текст, который можно вставить в новый чат, чтобы быстро восстановить контекст
- README.md — как запускать

Мини-план на твоём ПК (прямо сейчас)

У тебя уже ~31 GB свободно на C: — красавчик, это как будто Windows наконец перестала держать тебя за горло.

1. Запусти 01_scan.bat (посмотрим, кто следующий пожиратель)
2. 02_cleanup_dryrun.bat (посмотреть, что удалится)
3. Если всё ок — 03_cleanup_execute.bat
4. Если хочешь добить NVIDIA app кэш — 05_cleanup_nvidia_app_cache.bat (лучше от админа)
5. Когда решишься на "план Б" с Win11 — сначала 06_backup_browsers.bat + 07_collect_session_zip.bat

Если скинешь свежий scan_report_*.json из этой сборки — я точно скажу "вот эти 3 папки режь, эти не трогай". </>

< 2/2 >      ...