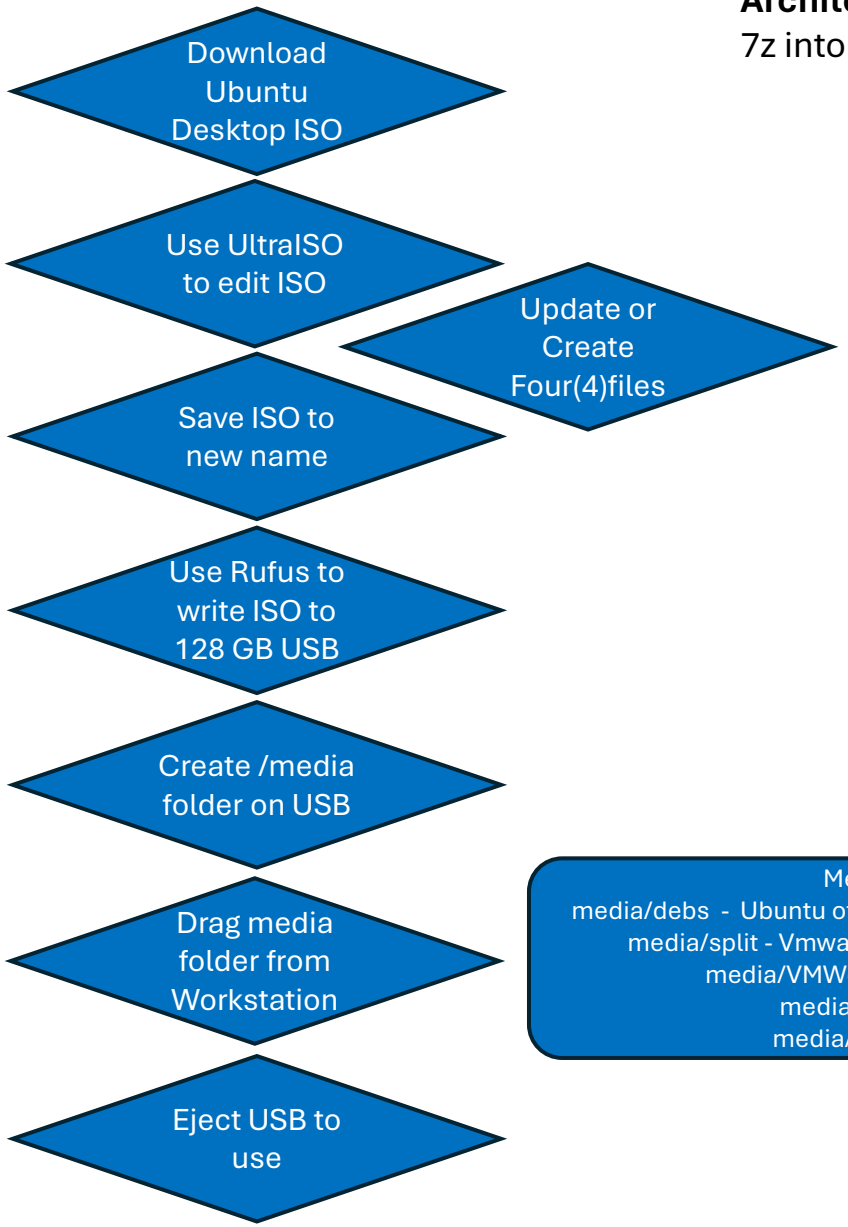
Goal: Kiosk USB Stick with Ubuntu OS & Vmware Workstation Pro and MS Win X Image

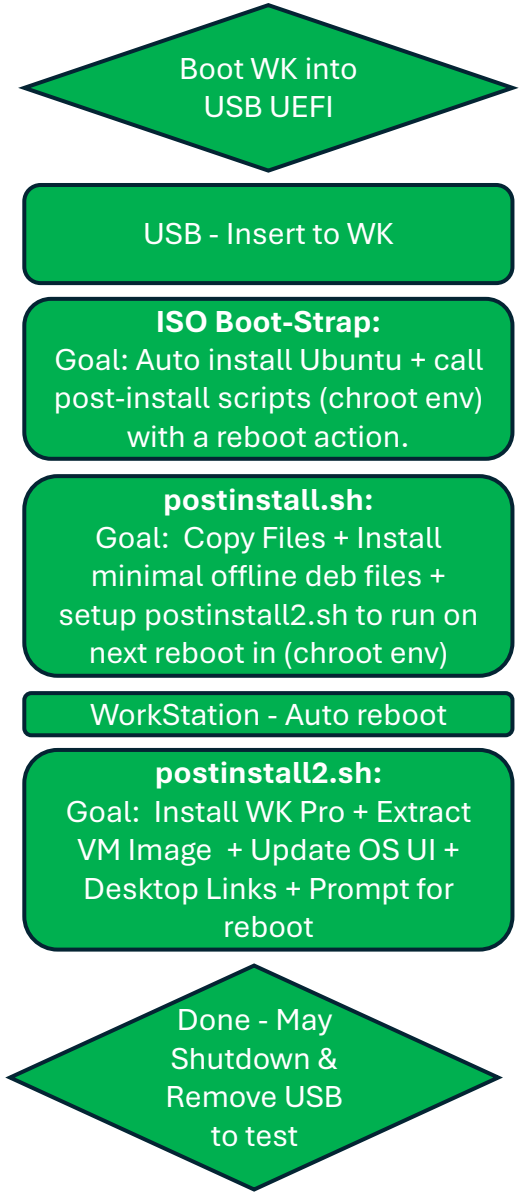
- Step 01: MS Win X image
  - Create Win10 Image with software as-needed.
  - Enable autologin with 'netplwiz' MS Win control panel tool. Reboot & Confirm.
  - Remove any vmware snapshots or older memory files (.vmem) in the current install folder.
  - Defrag & Compress Disk Image using VMware disk settings to smallest size.
  - Use 7zip to compress file & name "win10.7z" & spilt to 4GB files to avoid FAT32 challenge on USB stick.
- Step 02: Use UltraISO to edit Ubuntu ISO to create new bootable ISO with four (4) custom boot files.
  - /autoinstall/user-data & /autoinstall/meta-data [create these files ahead of time]
  - /boot/grub/grub.cfg & /boot/grub/loopback.cfg [update these files ahead of time]
- Step 03: Use Rufus with the new custom bootable ISO to a 64 GB USB stick.
- Step 04: Edit files on USB stick with Ubuntu
  - Create a folder /media & a shell script under /media/postinstall.sh [This can be edited on the fly as needed to change installation.]
- Step 05: Initial Validation of boot USB in a new system with auto-login (kiosk architecture)
  - Ubuntu should auto-install with no internet access. [May test with no files under /media - to ensure it can auto-install the os]
  - Ubuntu desktop UI should auto-login.
  - No other files are available, but we are looking for basic confirmation first.
- Step06: Copy the following install files to /media [These files will be installed by postinstall.sh & postinstall2.sh]
  - 7z2501-linux-x64.tar.xz
  - VMware-Workstation-Full-25H2-24995812.x86_64.bundle
  - win10.7z [the 10-20 4gb split files]
- Step07: Validate the boot USB in a new system with auto-login (kiosk architecture) with post-install process.
  - Ubuntu should auto-install with no internet access.
  - Ubuntu desktop UI should auto-login.
  - Vmware Pro UI should load with the MS Win 10 image
  - MS Win 10 image should auto-start and login to desktop with no interaction.

Architecture Model: 1 USB Stick and Win10 Image
7z into many files to fit FAT32 specs

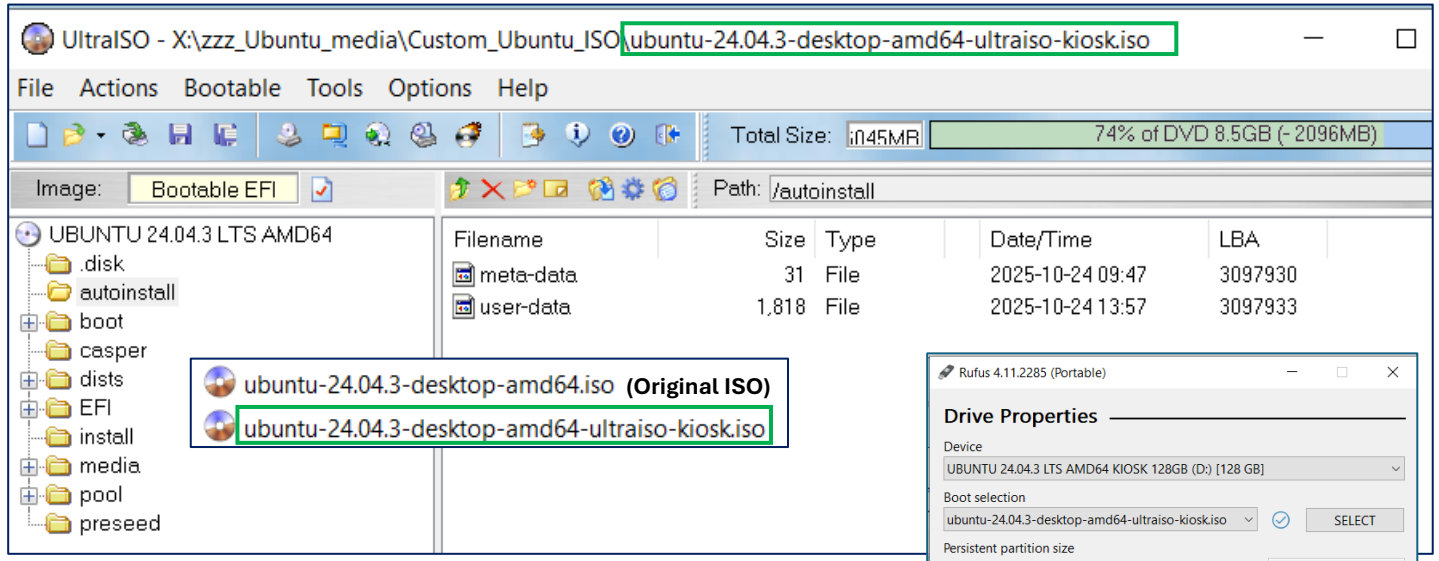


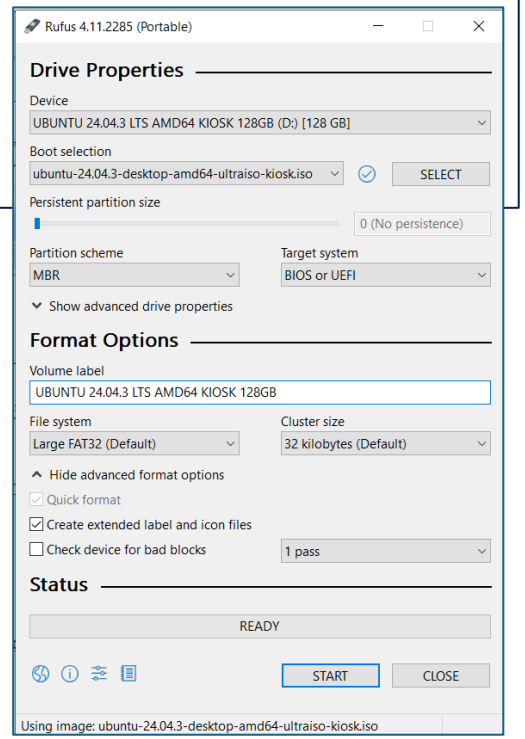
Media folder:
media/debs - Ubuntu offline deb install files including 7z
media/split - Vmware Image 7z split to 4GB size
media/VMWorkStationPro.bundle
media/postinstall.sh
media/postinstall2.sh

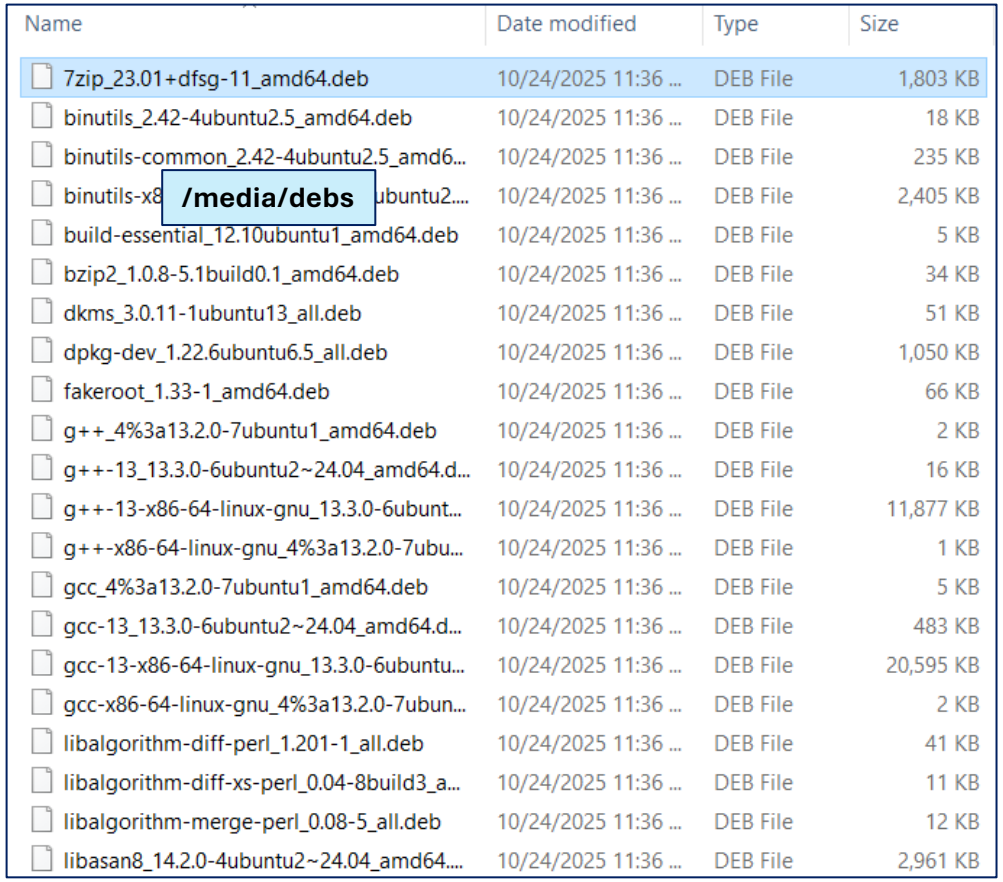
*** About 50 MB/s minimal speed for copy of compress 7z image, e.g. 15-20 min to copy in full. About 5-10 min to extract.



Process & Tools Used to create Ubuntu Kiosk USB







1. **Ubuntu Desktop ISO** (Latest/Standard with UI)
2. **UltraISO** (ISO editor)
   a. Edit the ISO's boot files
      a. /boot/grub/grub.cfg
      b. /boot/grub/loopback.cfg
      c. /autoinstall/meta-data
      d. /autoinstall/user-data [4 last-cmds]
3. **Rufus** (ISO->USB)
   a. Write ISO to USB Flash Drive w/ UEFI
4. **postinstall.sh** (Bash Shell Script in /media)
   a. Copy files from /media to SDD
   b. Install offline packages: /media/debs/(.deb files)
   c. Create SWAP file
   d. Auto reboot
5. **postinstall2.sh** [systemd one-shot upon 1st reboot]
   a. Install any other item we wish (Vmware Wk Pro)
   b. Update as needed.

Note: six (6) custom files for this architecture
- Four (4) embedded in the ISO image
- Two (2) external on USB under /media

```
UBUNTU 24_0 (D:)
├─ autoinstall
│  ├─ meta-data
│  └─ user-data
│
├─ boot
│  ├─ memtest86+x64.bin
│  └─ grub
│     ├─ grub.cfg
│     └─ loopback.cfg
│
└─ media
   ├─ postinstall.sh
   ├─ postinstall2.sh
   ├─ vmware-workstation-pro-25h2.pdf
   ├─ VMware-Workstation-Full-25H2-24995812.x86_64.bundle
   │
   ├─ debs   (offline packages)
   │  ├─ 7zip_23.01+dfsg-11_amd64.deb
   │  ├─ dkms_3.0.11-1ubuntu13_all.deb
   │  ├─ build-essential_12.10ubuntu1_amd64.deb
   │  ├─ linux-headers-6.8.0-86-generic_6.8.0-86.87_amd64.deb
   │  ├─ linux-headers-6.8.0-86_6.8.0-86.87_all.deb
   │  ├─ linux-headers-generic_6.8.0-86.87_amd64.deb
   │  ├─ gcc-13_13.3.0-6ubuntu2~24.04_amd64.deb
   │  ├─ g++-13_13.3.0-6ubuntu2~24.04_amd64.deb
   │  ├─ make_4.3-4.1build2_amd64.deb
   │  ├─ yamllint_1.33.0-1_all.deb
   │  └─ … (binutils, libs, notes.txt, etc.)
   │
   └─ split   (Windows VM archive parts)
      ├─ win10.7z.001
      ├─ win10.7z.002
      ├─ win10.7z.003
      ├─ win10.7z.004
      ├─ win10.7z.005
      ├─ win10.7z.006
      ├─ win10.7z.007
      ├─ win10.7z.008
      ├─ win10.7z.009
      ├─ win10.7z.010
      ├─ win10.7z.011
      └─ win10.7z.012
```

```
instance-id: ubuntu-autoinstall
```

```
#cloud-config
autoinstall:
  version: 1
  identity:
    hostname: ubuntu
    username: ubuntu
    # password: ubuntu (SHA-512 crypt)
    # ------------------------------------------------------------------
    # To generate a new SHA-512 password hash for this field:
    #  1. Boot any Linux system (including Ubuntu Live USB)
    #  2. Run this command in a terminal:
    #       mkpasswd -m sha-512
    #     or, if 'mkpasswd' isn't installed:
    #       python3 -c "import crypt; print(crypt.crypt(input('Password: '), crypt.mksalt(crypt.METHOD_SHA512)))"
    #  3. Copy the resulting string (it starts with $6$...) into the 'password:' line below.
    # ------------------------------------------------------------------
    password: "$6$rounds=4096$ubuntu$U9PnjptXDOpah3aHRjwM6e6r1kdDPmsrQoDyETvmE9ztvjDtdVjNIEF5hSrGD4D2Jv0Z/JOFGbtOa2h4f4u4c1"

  locale: en_US
  timezone: America/Chicago
  keyboard:
    layout: us

  ssh:
    install-server: false

  storage:
    layout:
      name: direct
    swap:
      size: 0          # prevent curtin swap bug; we create swap in postinstall
    overwrite: true

  packages: []         # offline: do not apt install anything during install
  user-data:
    disable_root: false
  late-commands:
    # 1) Copy + chmod inside the target (never fail curtin)
    - curtin in-target --target=/target -- bash -c 'cp -f /cdrom/media/postinstall.sh /root/postinstall.sh && chmod +x /root/postinstall.sh || true'

    # 2) Run Phase 1 work INSIDE target, but never fail curtin (log rc)
    - curtin in-target --target=/target -- bash -c 'set +e; /root/postinstall.sh > /var/log/postinstall.log 2>&1; echo "[INFO] postinstall.sh (in-target) rc=$?" >> /var/log/postinstall.log; exit 0'

    # 3) Schedule reboot from the LIVE environment (outside chroot)
    #    Release /target so Subiquity can unmount cleanly; detach reboot.
    - bash -c 'cd /; fuser -km /target || true; umount -R /target 2>/dev/null || umount -lR /target 2>/dev/null || true; (nohup /sbin/reboot -f >/dev/null 2>&1 <&- &) ; exit 0'

    # 4) Failsafe: ensure GDM autologin even if postinstall.sh didn't run
    - curtin in-target --target=/target -- bash -c "mkdir -p /etc/gdm3 && printf '[daemon]\nAutomaticLoginEnable=true\nAutomaticLogin=ubuntu\n' > /etc/gdm3/custom.conf || true"
```

```
set default=0
set timeout=10
set timeout_style=menu

loadfont unicode

set menu_color_normal=white/black
set menu_color_highlight=black/light-gray

# Optional: show a warning banner on the menu
insmod gfxterm
terminal_output gfxterm
echo
echo '*** WARNING: AUTOINSTALL WILL ERASE THE INTERNAL DISK ***'
echo 'Use "NO AUTOINSTALL" if you do NOT want to wipe the disk.'
echo 'Press e on an entry to edit and remove autoinstall if needed.'
echo
sleep 3

# Autoinstall (destructive)
# Ensure the USB has /autoinstall/user-data and /autoinstall/meta-data.
# Keep your helpers in /media/ (postinstall.sh, VMware bundle, 7-Zip tar, win10.7z)
menuentry "Auto Install Ubuntu with VMware Workstation Pro in Kiosk Mode" {
    echo 'Autoinstall starting… internal disk will be overwritten.'
    sleep 3
    set gfxpayload=keep
    linux   /casper/vmlinuz autoinstall 'ds=nocloud;s=/cdrom/autoinstall' quiet text ---
#   linux   /casper/vmlinuz autoinstall 'ds=nocloud;s=/cdrom/autoinstall' quiet splash ---
    initrd  /casper/initrd
}

```

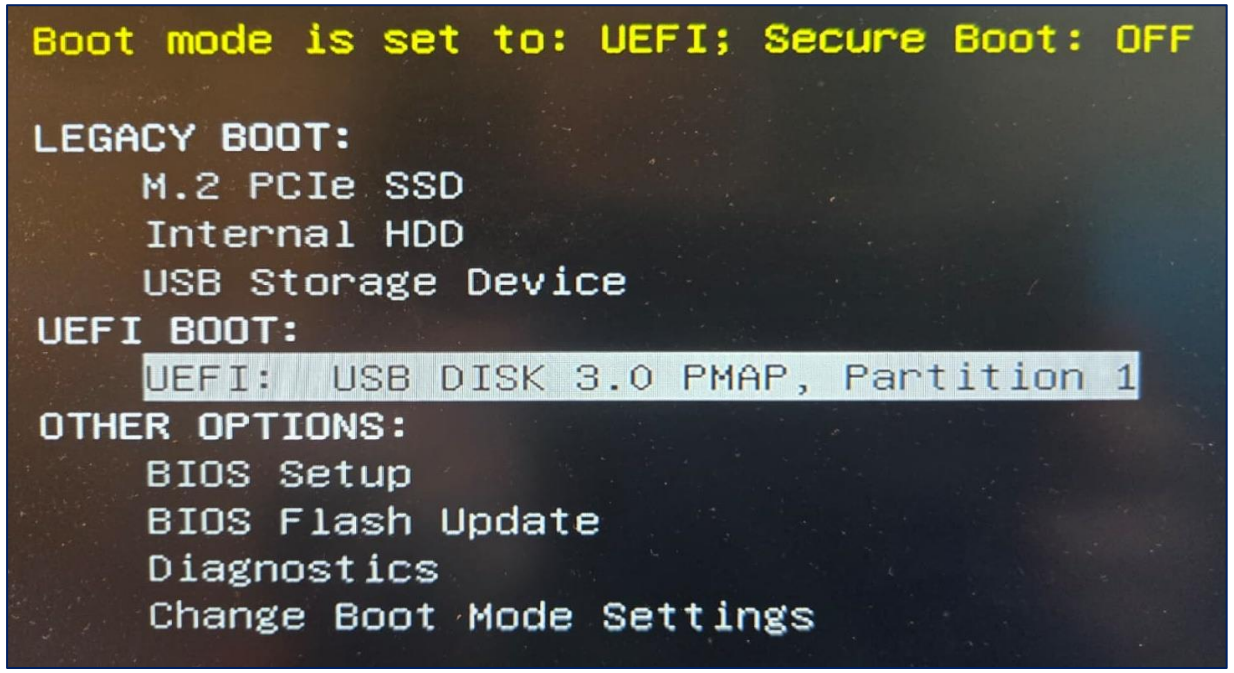
```

menuentry "Kiosk Ubuntu AutoInstall" {
    set gfxpayload=keep
    linux   /casper/vmlinuz autoinstall 'ds=nocloud;s=/cdrom/autoinstall' quiet text ---
#   linux   /casper/vmlinuz autoinstall 'ds=nocloud;s=/cdrom/autoinstall' quiet splash ---
    initrd  /casper/initrd
}
menuentry "Ubuntu (safe graphics)" {
    set gfxpayload=keep
    linux   /casper/vmlinuz nomodeset autoinstall 'ds=nocloud;s=/cdrom/autoinstall' quiet splash text ---
    initrd  /casper/initrd
}
```

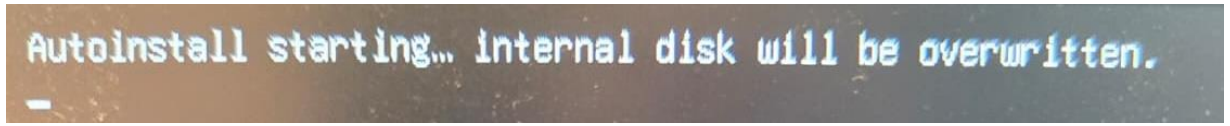
Four (4) boot-strap ISO configuration files:
Goals: Install the OS then run a primary post-install script (that runs in chroot env) & reboots.

Data Flow:

- UEFI will load grub.cfg / loopback.cfg first.
- These files will load the /autoinstaller folder.
- The "meta-data" and "user-data" will be referenced.
- The user-data's "late-commands" will then call "postinstall.sh" script.
    - The "user-data" file must be a YAML / UTF-8 / UNIX LF Return format.
- postinstall.sh (chroot env) will set a few configurations, then set postinstall2.sh to run on next reboot.
- postinstall2.sh will install software and desktop links

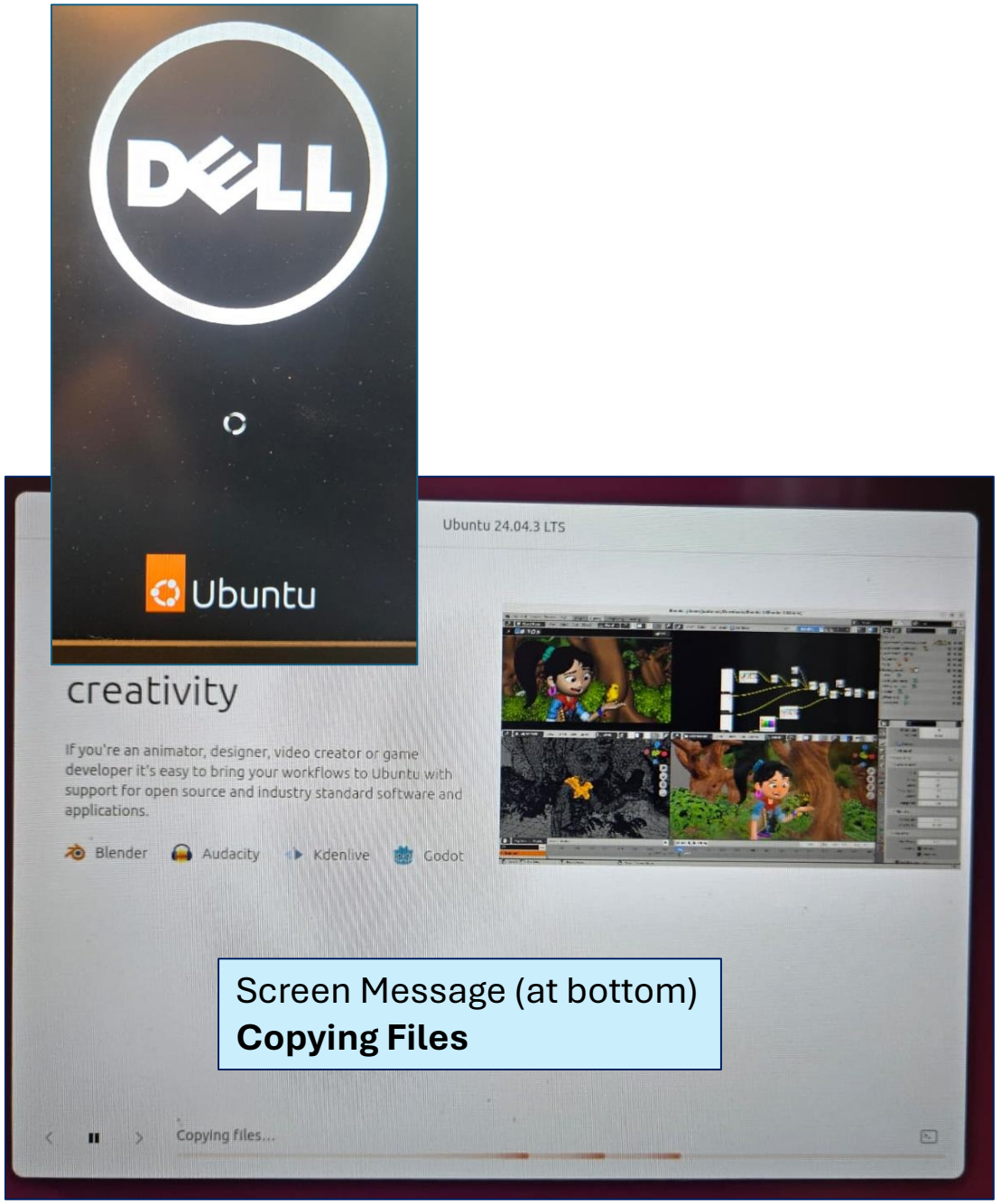
Boot USB via UEFI





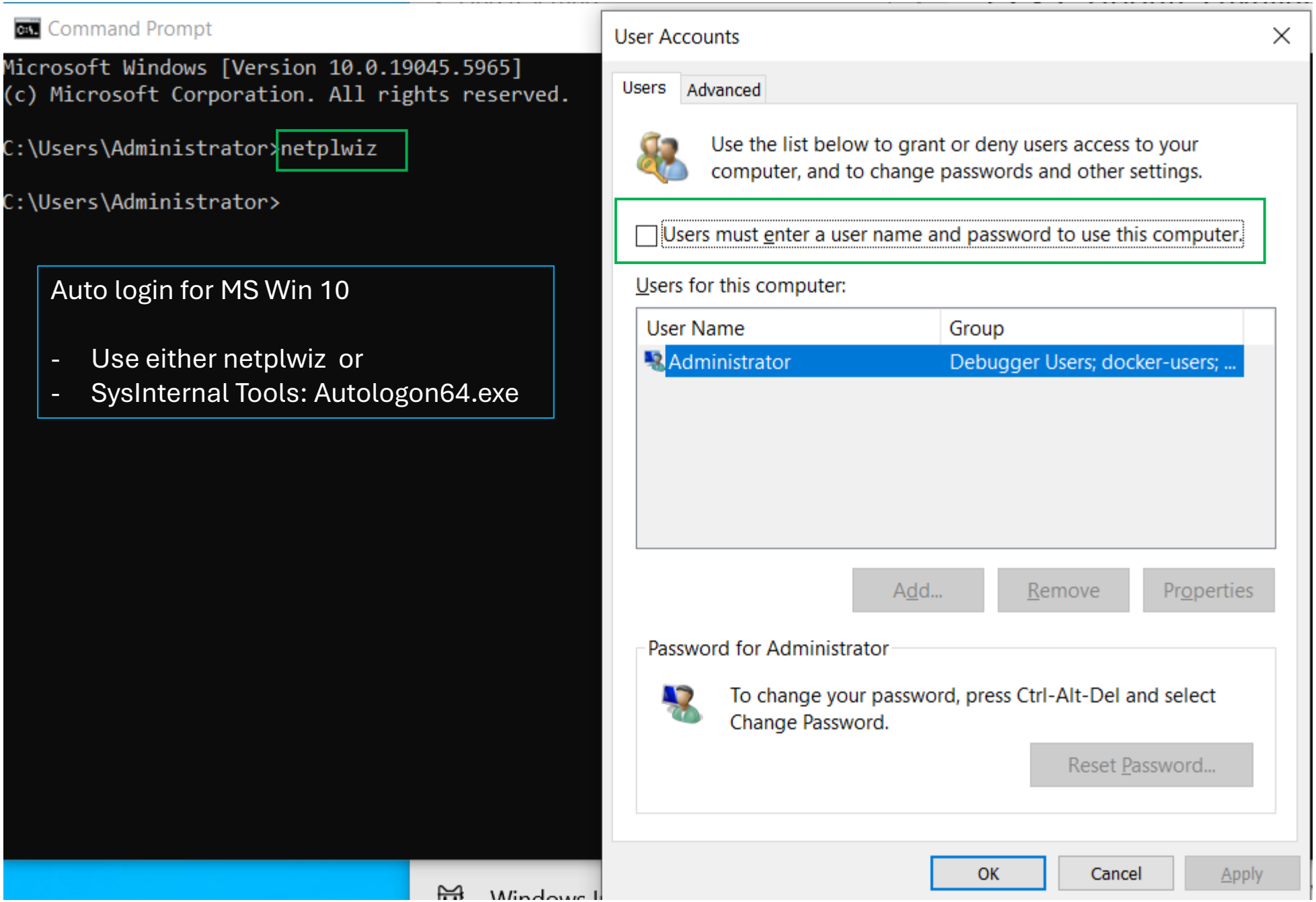
Timing: Dell Precision 7510 Laptop

- 1st boot: 12-15 min from UEFI Boot, to the installation of Ubuntu OS & install of offline deb files & copy of /media folder & auto-reboot.
- 2nd reboot: 15-20 min, install Vmware Wk pro & extract Vmware image, configurations & request for reboot.
- 3rd reboot: 1-3 min, auto load Vmware WK image.



Screen Message (at bottom)
Copying Files

Prepare the MS Windows Vmware Image

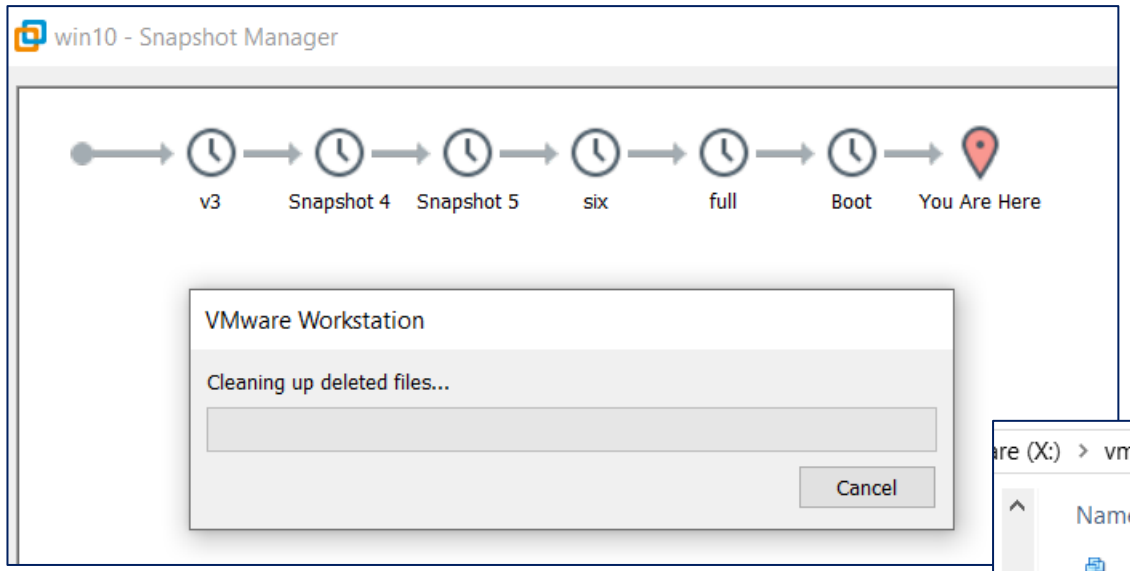


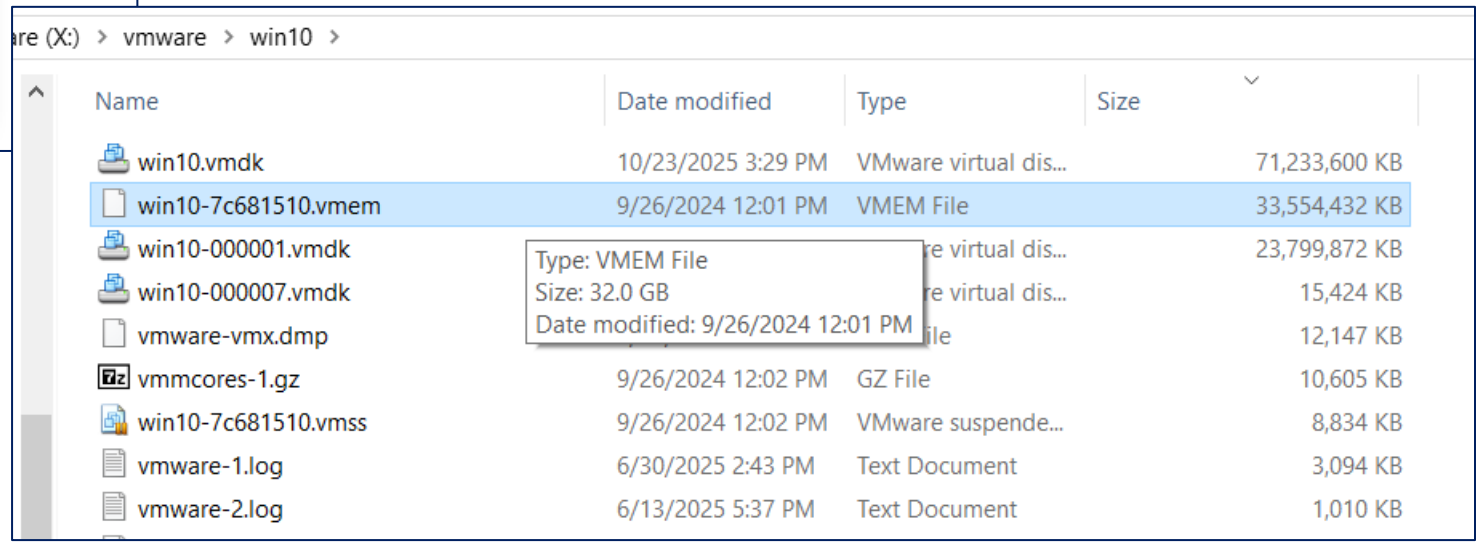
Auto login for MS Win 10

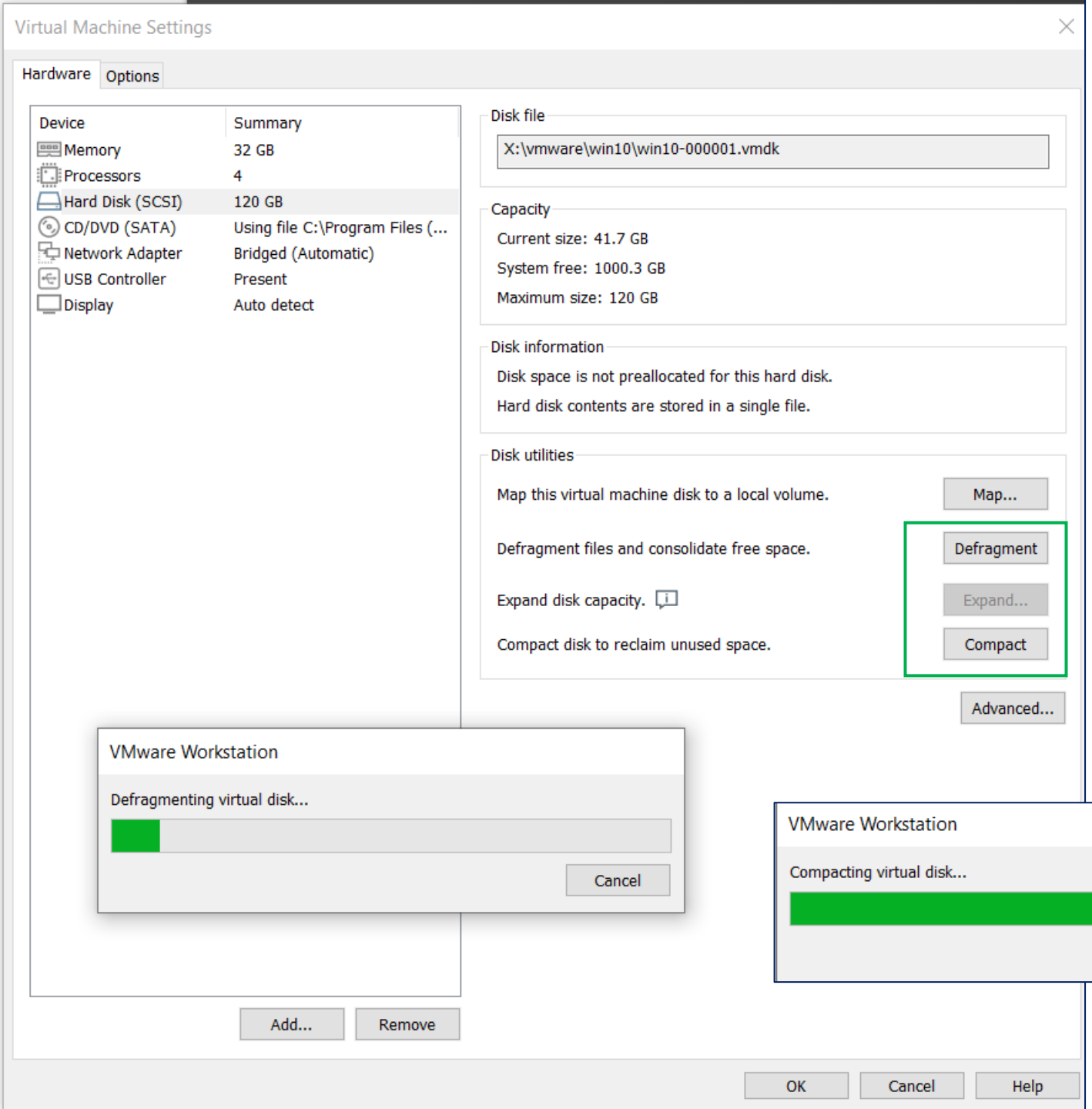
- Use either netplwiz or
- SysInternal Tools: Autologon64.exe

To reduce size:

- If there are any VMware SNAPSHOTS, please delete them prior to defrag/compression efforts.
  - Optionally: Clone current version to a new deployment with no snapshots.
- Delete any "old" suspended memory files (.vmem) (back up prior to any deletion)



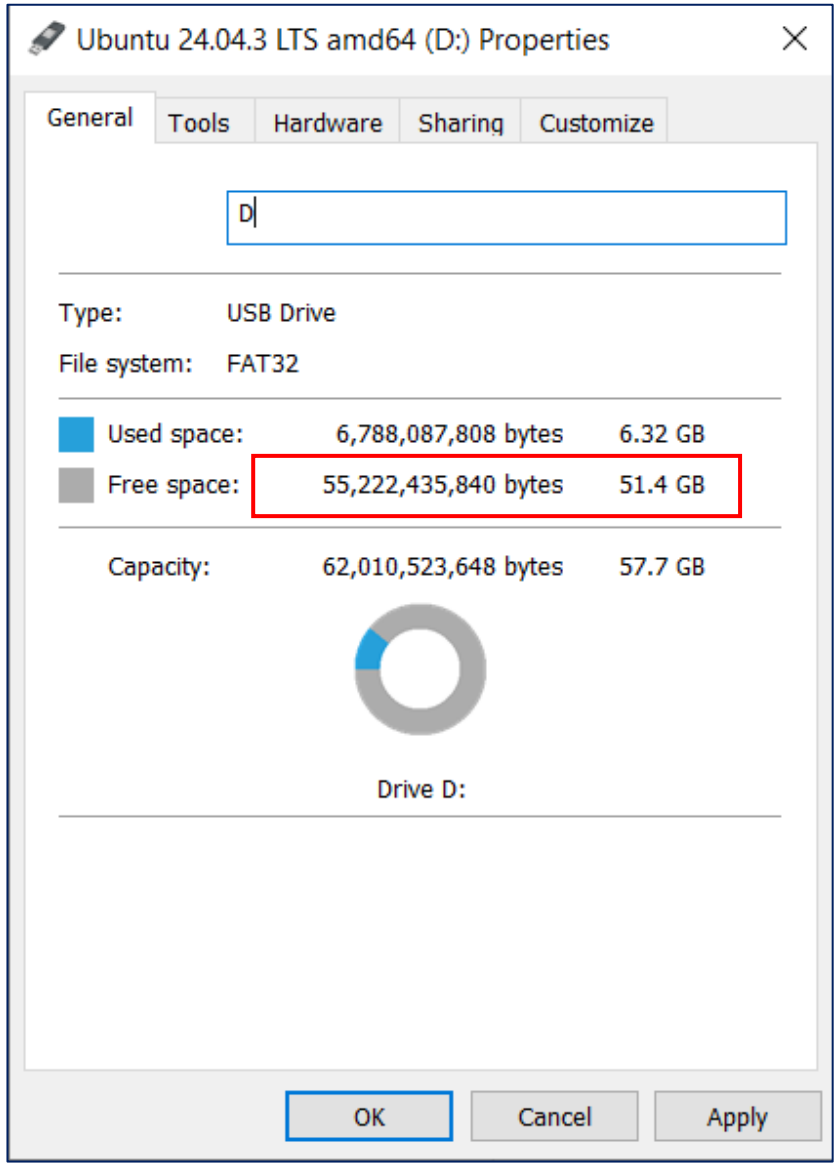


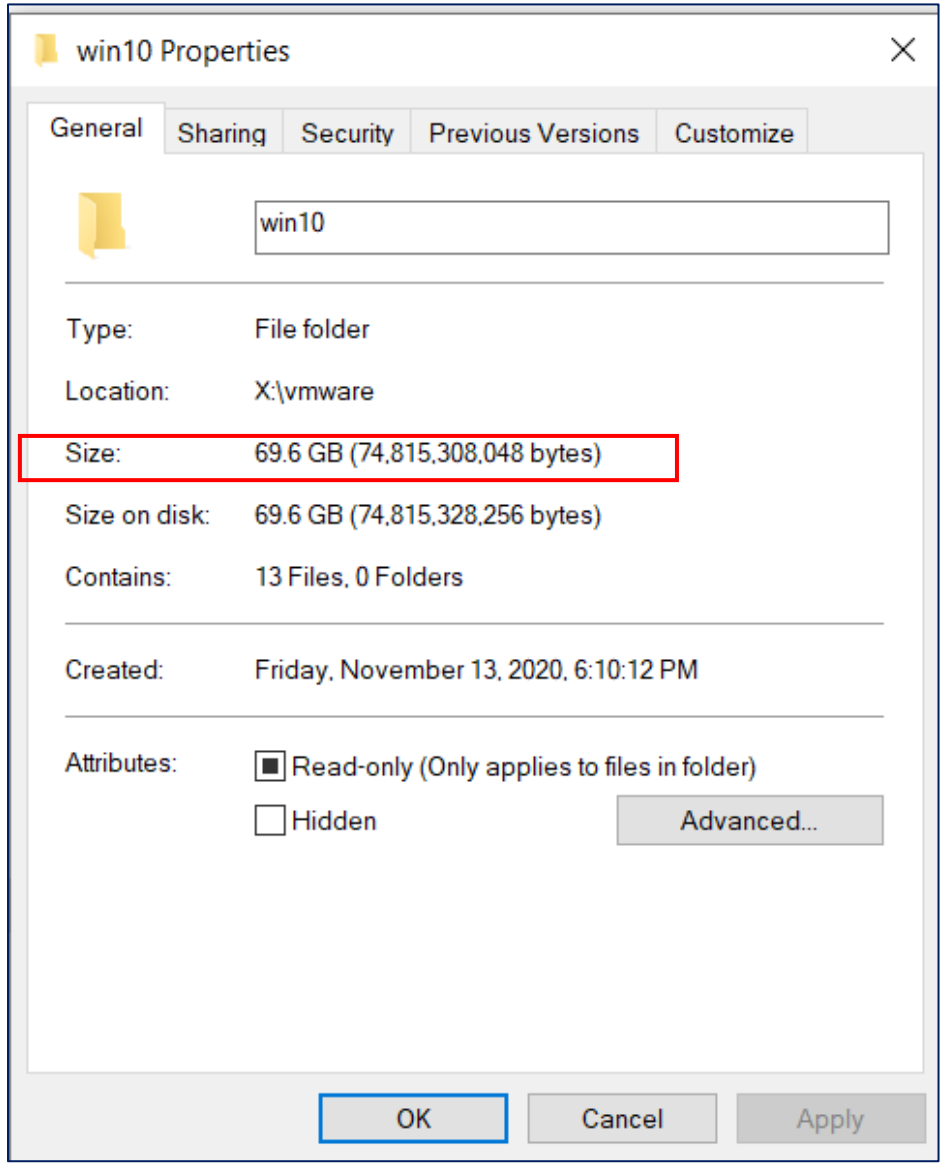


Defrag and Compress the Hard Drive for Vmware Image to fit on USB stick

- We are aiming for a compressed image folder less than 50 GB

Ubuntu UBS 64 GB Boot has 51 GB free space.
Current Vmware Image Folder size is greater.

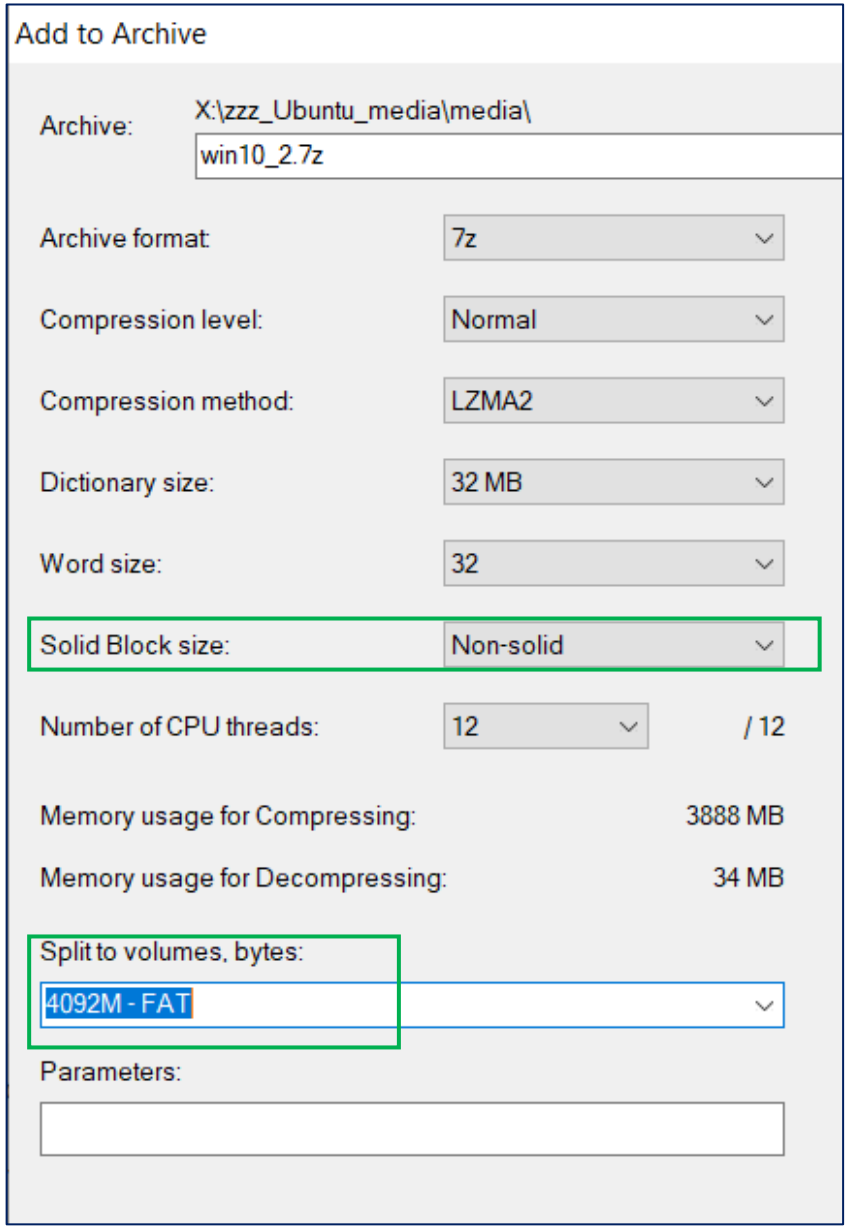


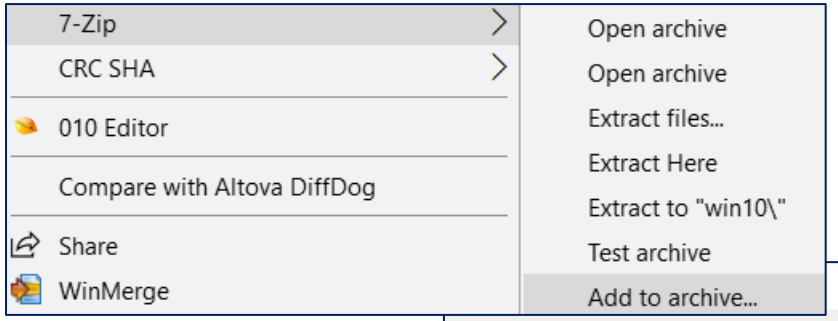


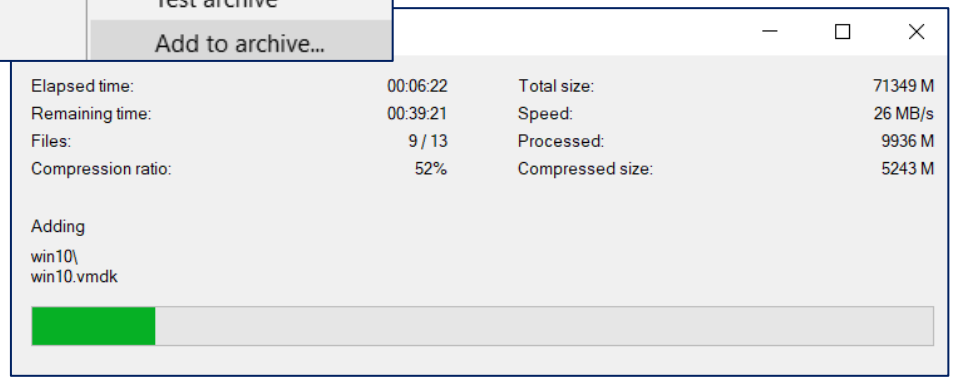
7zip notes

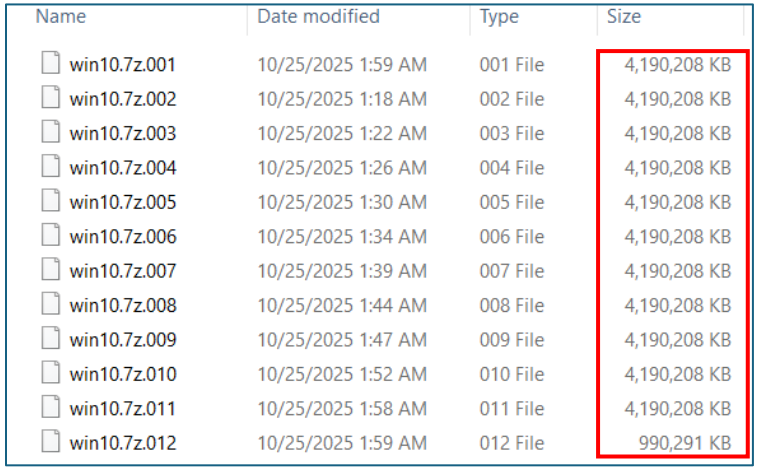
Use 7zip & adjust settings -

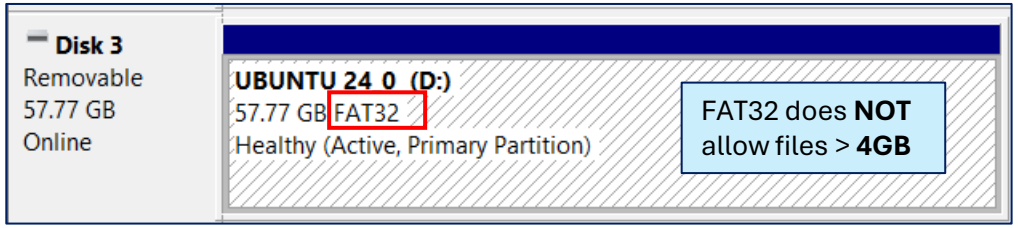
- Select "**Non-Solid**" for speed on extraction
- Spilt to **4 GB** Files (to avoid FAT32 challenge)





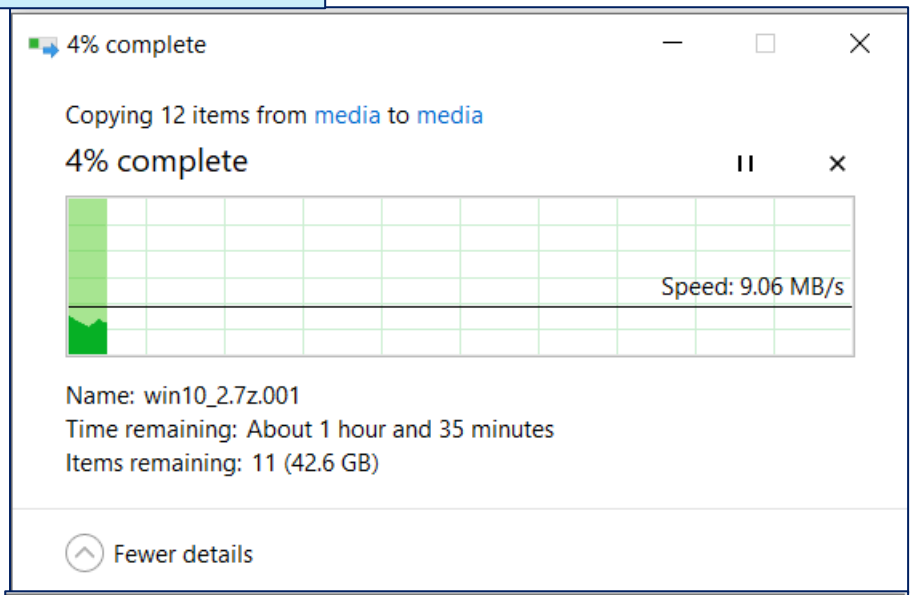


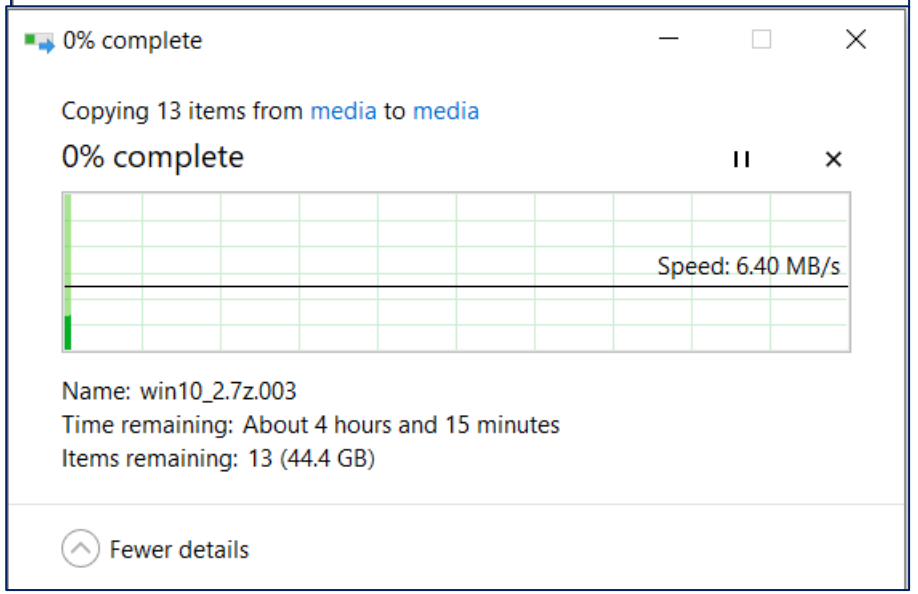




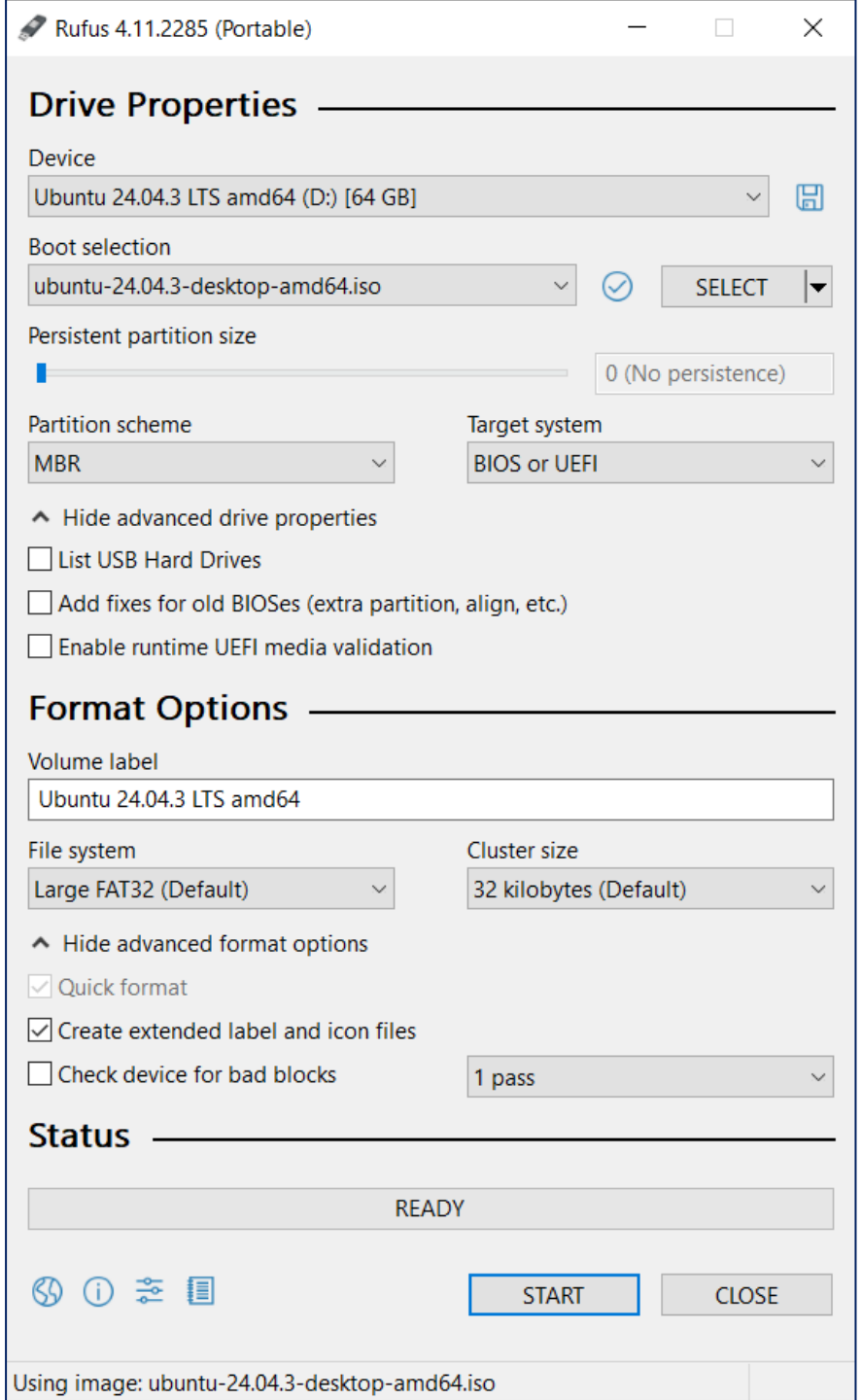
Final media folder state & test of media copy process speed.
- Use USB specs 3.1/3.2 for performance for the USB stick.

Name	Date modified	Type	Size
7z2501-linux-x64.tar.xz	10/23/2025 1:31 PM	XZ File	1,535 KB
gueststore-vmtools-13.0.5-0.24916190.tar.gz	10/23/2025 1:29 PM	GZ File	100,057 KB
VMware-Workstation-Full-25H2-24995812.x86_64.bundle	10/23/2025 1:28 PM	BUNDLE File	302,273 KB
win10_2.7z.001	10/23/2025 5:16 PM	001 File	4,190,208 KB
win10_2.7z.002	10/23/2025 4:48 PM	002 File	4,190,208 KB
win10_2.7z.003	10/23/2025 4:51 PM	003 File	4,190,208 KB
win10_2.7z.004	10/23/2025 4:54 PM	004 File	4,190,208 KB
win10_2.7z.005	10/23/2025 4:57 PM	005 File	4,190,208 KB
win10_2.7z.006	10/23/2025 5:00 PM	006 File	4,190,208 KB
win10_2.7z.007	10/23/2025 5:03 PM	007 File	4,190,208 KB
win10_2.7z.008	10/23/2025 5:07 PM	008 File	4,190,208 KB
win10_2.7z.009	10/23/2025 5:10 PM	009 File	4,190,208 KB
win10_2.7z.010	10/23/2025 5:13 PM	010 File	4,190,208 KB
win10_2.7z.011	10/23/2025 5:16 PM	011 File	4,190,208 KB
win10_2.7z.012	10/23/2025 5:16 PM	012 File	935,489 KB





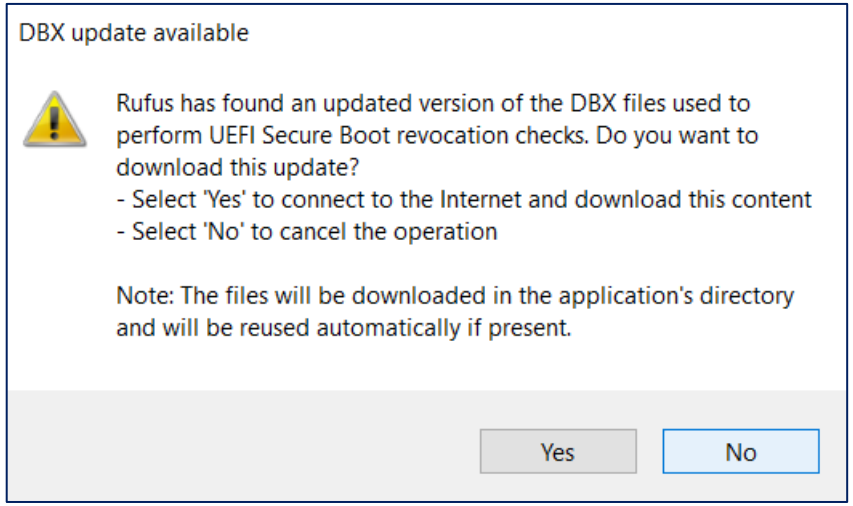
Rufus notes

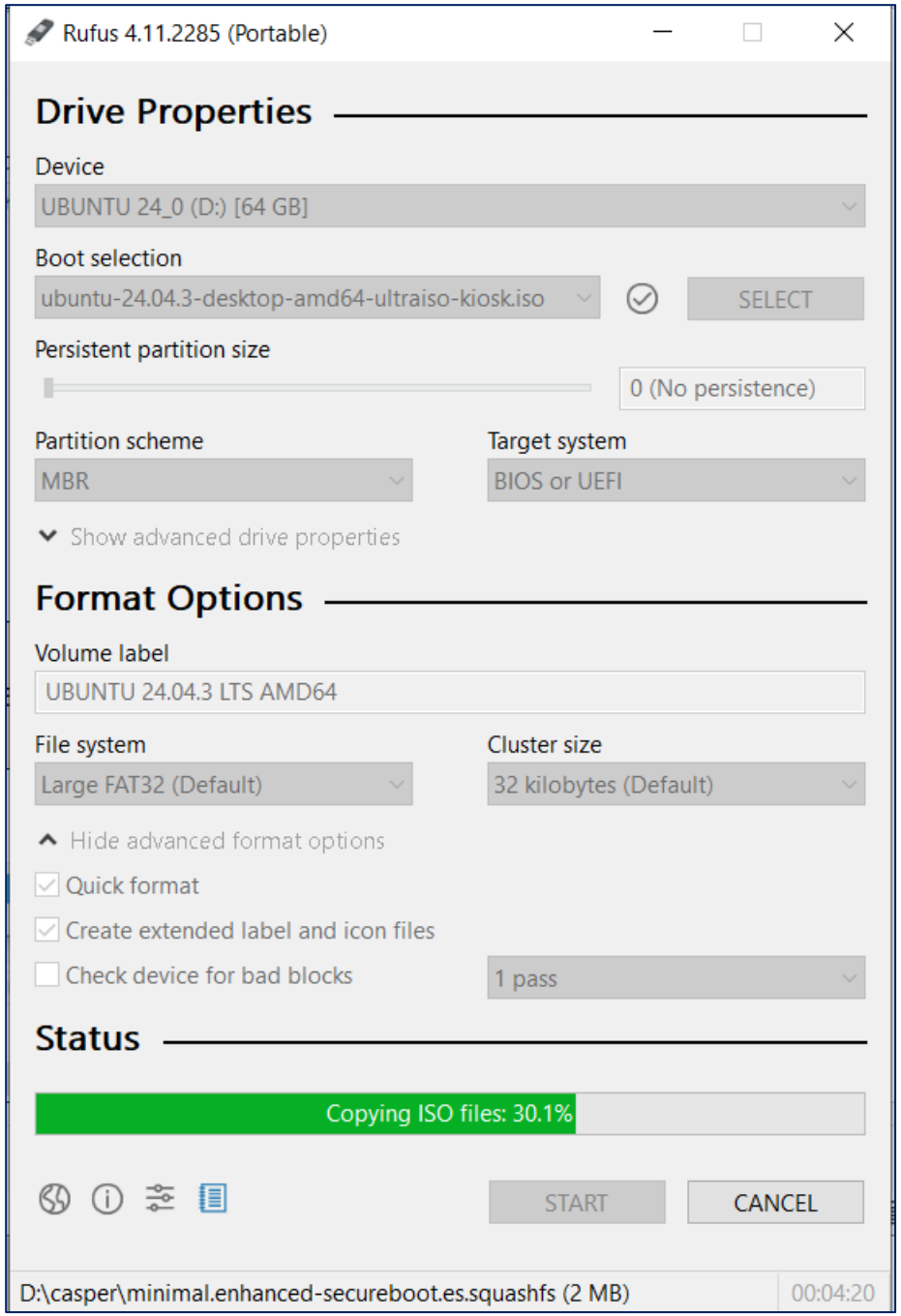


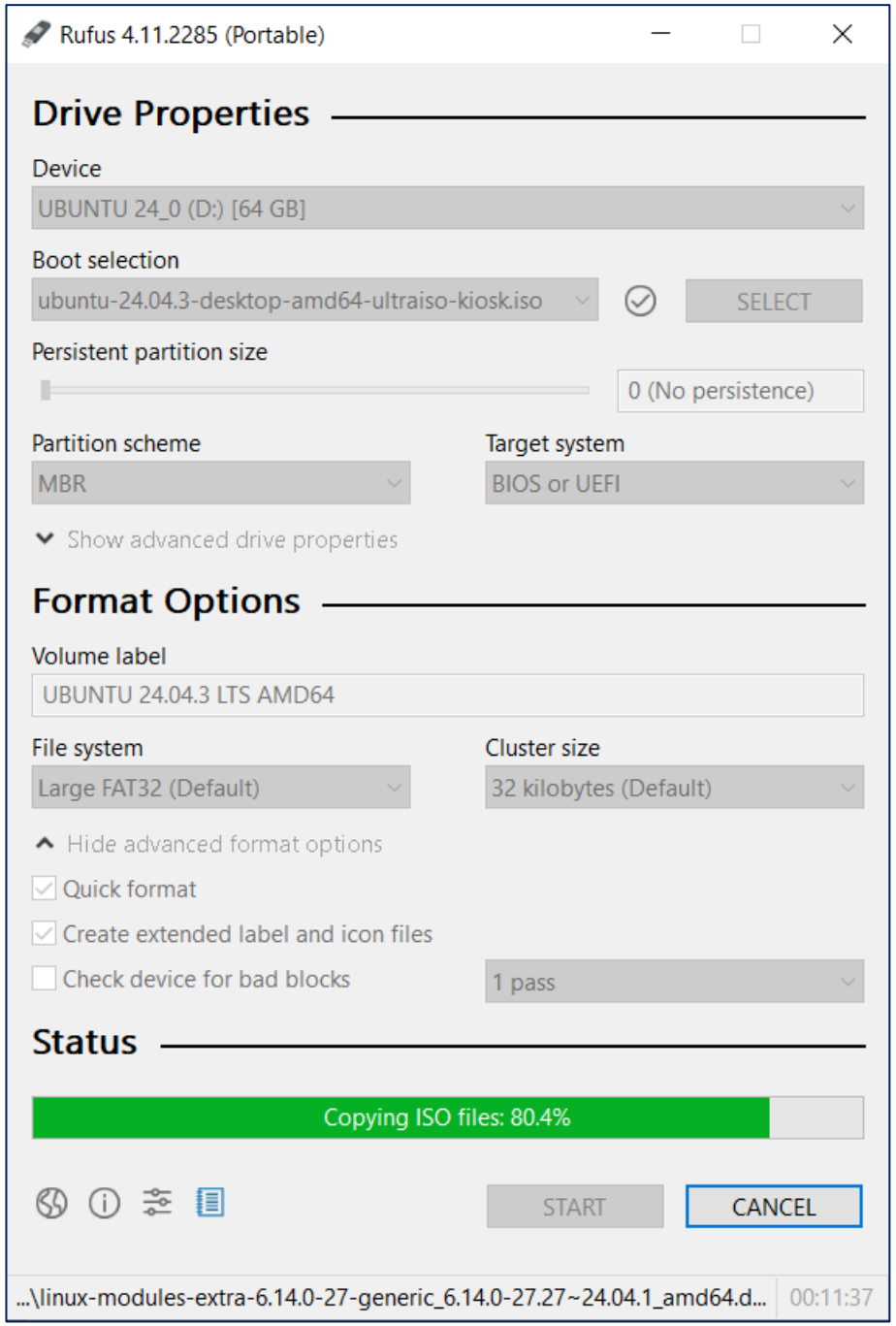
Use Rufus executable to create bootable USB stick.

- Rufus allows NTFS file system type, but it is not 100% for all newer BIOS UEFI to boot correctly, so we selected standard FAT32 file system.

- Avoid updating the DBX files for "secure boot" - not required for the kiosk architecture.

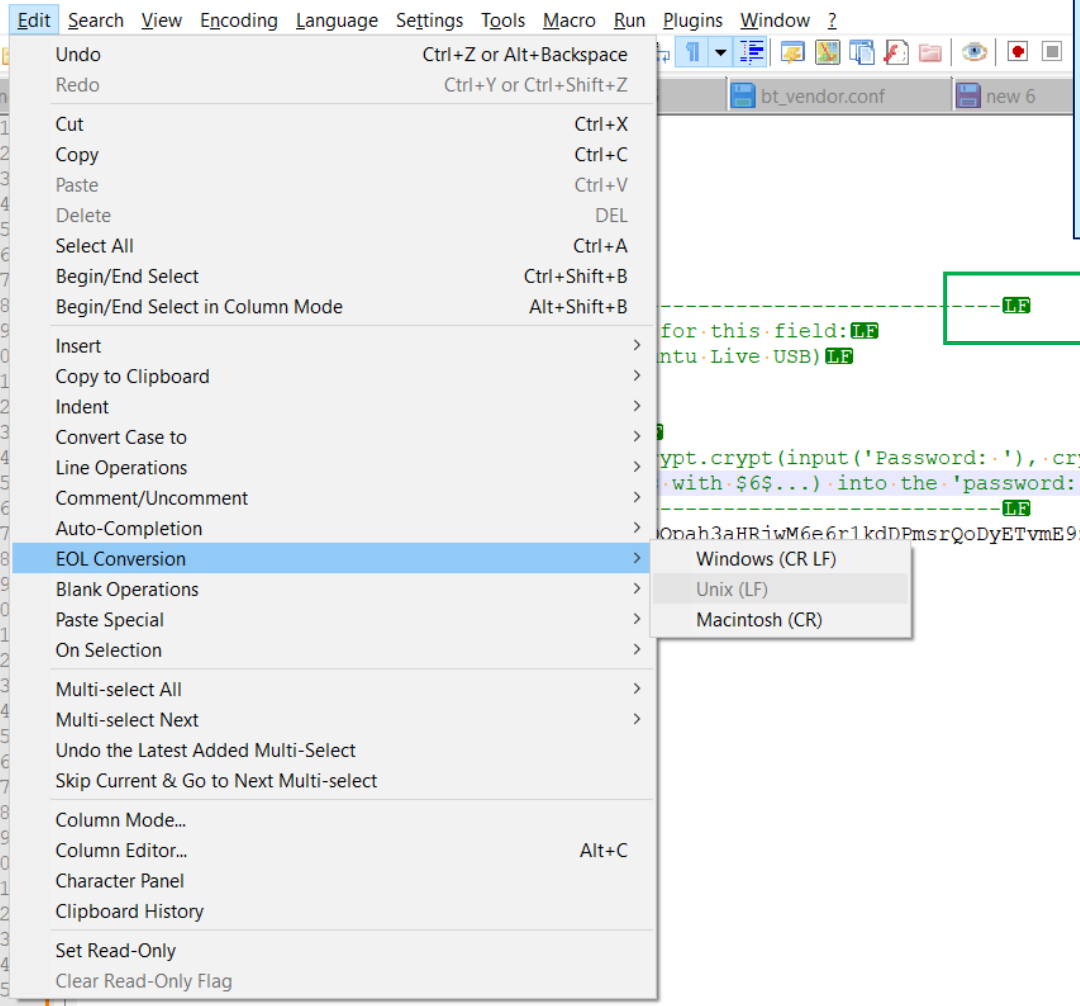


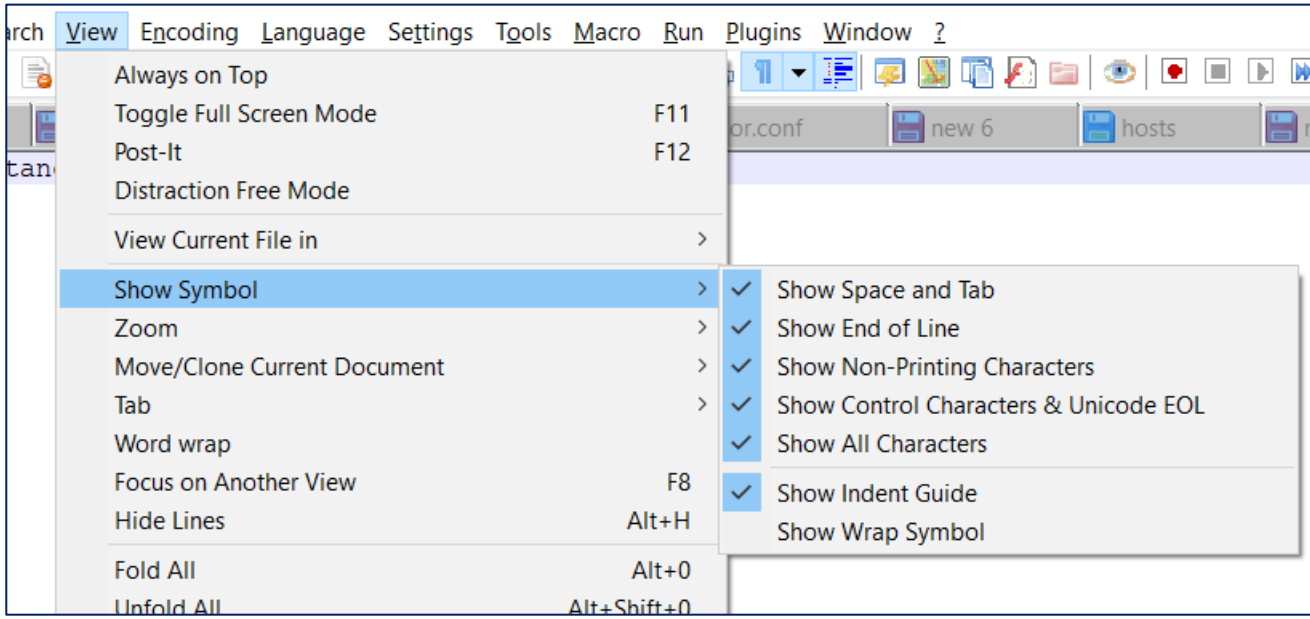

Rufus 4.11.2285 (Portable)
Drive Properties
Device
UBUNTU 24_0 (D:) [64 GB]
Boot selection
ubuntu-24.04.3-desktop-amd64-ultraiso-kiosk.iso
SELECT
Persistent partition size
0 (No persistence)
Partition scheme
MBR
Target system
BIOS or UEFI
Show advanced drive properties
Format Options
Volume label
UBUNTU 24.04.3 LTS AMD64
File system
Large FAT32 (Default)
Cluster size
32 kilobytes (Default)
Hide advanced format options
Quick format
Create extended label and icon files
Check device for bad blocks
1 pass
Status
Copying ISO files: 30.1%
START
CANCEL
D:\casper\minimal.enhanced-secureboot.es.squashfs (2 MB)
00:04:20

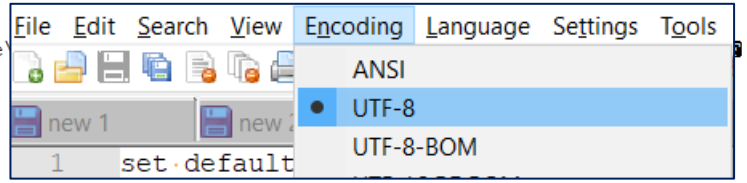

Rufus 4.11.2285 (Portable)
Drive Properties
Device
UBUNTU 24_0 (D:) [64 GB]
Boot selection
ubuntu-24.04.3-desktop-amd64-ultraiso-kiosk.iso
SELECT
Persistent partition size
0 (No persistence)
Partition scheme
MBR
Target system
BIOS or UEFI
Show advanced drive properties
Format Options
Volume label
UBUNTU 24.04.3 LTS AMD64
File system
Large FAT32 (Default)
Cluster size
32 kilobytes (Default)
Hide advanced format options
Quick format
Create extended label and icon files
Check device for bad blocks
1 pass
Status
Copying ISO files: 80.4%
START
CANCEL
...\linux-modules-extra-6.14.0-27-generic_6.14.0-27.27~24.04.1_amd64.d...
00:11:37

Format Notes

Ensure all files are UNIX LF (line feed) ending characters.
- Use View / Show Symbol / Select Space & Tab & Show End of Line
- Use Edit / EOL Conversion / Unix (LF) to convert.
- Validate the Encoding is set to: UTF-8 Encoding w/o BOM





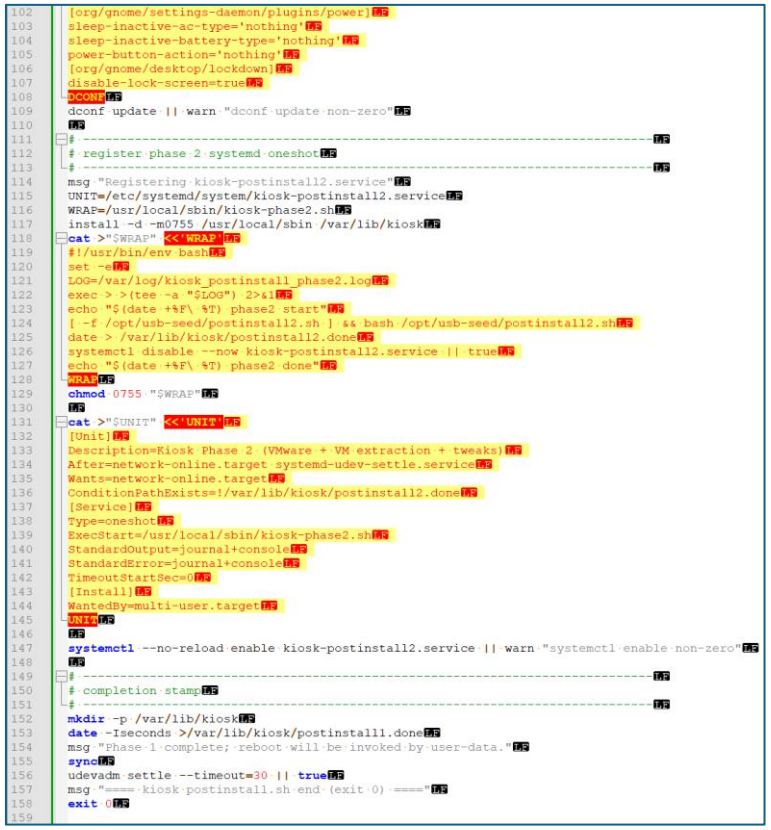


postinstall.sh & postinstall2.sh

postinstall.sh (runs in a chroot env - so we can only do selected updates)

```
#!/usr/bin/env bash
# ============================================================================
# kiosk postinstall.sh — Phase 1 (always executed inside curtin chroot)
#  - Stages offline media to /opt/usb-seed
#  - Installs any .deb payloads (best-effort)
#  - Creates swapfile
#  - Enables GDM autologin + disables lock screen
#  - Registers Phase 2 systemd oneshot
#  - Always exits 0 so Subiquity never fails (reboot handled by user-data)
# ============================================================================
VERSION=1.3.3
BUILD_DATE=2025-10-29

# relaxed error handling inside curtin chroot
set +e +o pipefail
umask 022
set +m
trap - ERR  # disable ERR trap completely

LOG=/var/log/kiosk_postinstall_phase1.log
exec > >(tee -a "$LOG") 2>&1
_ts() { date +"%F %T"; }
msg() { printf '[%s] %s\n' "$(_ts)" "$*"; }
warn() { printf '[%s] [WARN] %s\n' "$(_ts)" "$*"; }

msg "==== kiosk postinstall.sh start (v${VERSION} ${BUILD_DATE}) ===="

# ----------------------------------------------------------------------------
# basic config
# ----------------------------------------------------------------------------
USB_MEDIA=${USB_MEDIA:-/cdrom/media}
SEED_DST=${SEED_DST:-/opt/usb-seed}
DEB_DIR=${DEB_DIR:-${SEED_DST}/debs}
USER_NAME=${USER_NAME:-ubuntu}
SWAP_GB=${SWAP_GB:-8}

for v in USB_MEDIA SEED_DST DEB_DIR USER_NAME SWAP_GB; do
  eval "msg \"$v=\${$v}\""
done

# ----------------------------------------------------------------------------
# stage offline media
# ----------------------------------------------------------------------------
msg "Staging offline media from $USB_MEDIA → $SEED_DST"
install -d -m 0755 "$SEED_DST"
rsync -avh --delete --info=NAME,STATS "$USB_MEDIA/" "$SEED_DST/" 2>&1
if [[ $? -ne 0 ]]; then
  warn "rsync failed, trying cp -a fallback"
  cp -a "$USB_MEDIA/." "$SEED_DST/" || warn "copy fallback failed"
fi
ls -la "$SEED_DST" || true

# ----------------------------------------------------------------------------
# install offline .debs (best effort)
# ----------------------------------------------------------------------------
if compgen -G "${DEB_DIR}/*.deb" >/dev/null 2>&1; then
  msg "Installing .debs from $DEB_DIR"
  apt-get update || warn "apt-get update failed (expected offline)"
  dpkg -i "${DEB_DIR}"/*.deb || warn "dpkg -i non-zero"
  apt-get -f install -y || warn "apt-get -f install non-zero"
else
  msg "No .deb payloads under $DEB_DIR"
fi

# ----------------------------------------------------------------------------
# create swapfile
# ----------------------------------------------------------------------------
msg "Creating ${SWAP_GB} GB swapfile"
SWAPFILE=/swapfile
if ! grep -q 'swapfile' /etc/fstab 2>/dev/null; then
  fallocate -l "${SWAP_GB}G" "$SWAPFILE" 2>/dev/null || \
    dd if=/dev/zero of="$SWAPFILE" bs=1M count=$((SWAP_GB*1024)) status=progress
  chmod 600 "$SWAPFILE"
  mkswap "$SWAPFILE" && swapon "$SWAPFILE"
  echo '/swapfile none swap sw 0 0' >> /etc/fstab
else
  msg "Swapfile already present"
fi
swapon --show || true

# ----------------------------------------------------------------------------
# GDM autologin
# ----------------------------------------------------------------------------
msg "Configuring GDM autologin for ${USER_NAME}"
install -d -m 0755 /etc/gdm3
cat >/etc/gdm3/custom.conf <<EOF
[daemon]
AutomaticLoginEnable=true
AutomaticLogin=${USER_NAME}
EOF

# ----------------------------------------------------------------------------
# disable lock screen / idle
# ----------------------------------------------------------------------------
msg "Writing dconf defaults"
install -d -m 0755 /etc/dconf/db/local.d
cat >/etc/dconf/db/local.d/00-kiosk <<'DCONF'
[org/gnome/desktop/screensaver]
lock-enabled=false
[org/gnome/desktop/session]
idle-delay=uint32 0
[org/gnome/settings-daemon/plugins/power]
sleep-inactive-ac-type='nothing'
sleep-inactive-battery-type='nothing'
power-button-action='nothing'
[org/gnome/desktop/lockdown]
disable-lock-screen=true
DCONF
dconf update || warn "dconf update non-zero"

# ----------------------------------------------------------------------------
# register phase 2 systemd oneshot
# ----------------------------------------------------------------------------
msg "Registering kiosk-postinstall2.service"
UNIT=/etc/systemd/system/kiosk-postinstall2.service
```

```
# ----------------------------------------------------------------------------
# create swapfile
# ----------------------------------------------------------------------------
msg "Creating ${SWAP_GB} GB swapfile"
SWAPFILE=/swapfile
if ! grep -q 'swapfile' /etc/fstab 2>/dev/null; then
  fallocate -l "${SWAP_GB}G" "$SWAPFILE" 2>/dev/null || \
    dd if=/dev/zero of="$SWAPFILE" bs=1M count=$((SWAP_GB*1024)) status=progress
  chmod 600 "$SWAPFILE"
  mkswap "$SWAPFILE" && swapon "$SWAPFILE"
  echo '/swapfile none swap sw 0 0' >> /etc/fstab
else
  msg "Swapfile already present"
fi
swapon --show || true

# ----------------------------------------------------------------------------
# GDM autologin
# ----------------------------------------------------------------------------
msg "Configuring GDM autologin for ${USER_NAME}"
install -d -m 0755 /etc/gdm3
cat >/etc/gdm3/custom.conf <<EOF
[daemon]
AutomaticLoginEnable=true
AutomaticLogin=${USER_NAME}
EOF

# ----------------------------------------------------------------------------
# disable lock screen / idle
# ----------------------------------------------------------------------------
msg "Writing dconf defaults"
install -d -m 0755 /etc/dconf/db/local.d
cat >/etc/dconf/db/local.d/00-kiosk <<'DCONF'
[org/gnome/desktop/screensaver]
lock-enabled=false
[org/gnome/desktop/session]
idle-delay=uint32 0
[org/gnome/settings-daemon/plugins/power]
sleep-inactive-ac-type='nothing'
sleep-inactive-battery-type='nothing'
power-button-action='nothing'
[org/gnome/desktop/lockdown]
disable-lock-screen=true
DCONF
dconf update || warn "dconf update non-zero"

# ----------------------------------------------------------------------------
# register phase 2 systemd oneshot
# ----------------------------------------------------------------------------
msg "Registering kiosk-postinstall2.service"
UNIT=/etc/systemd/system/kiosk-postinstall2.service
```



```
[org/gnome/settings-daemon/plugins/power]
sleep-inactive-ac-type='nothing'
sleep-inactive-battery-type='nothing'
power-button-action='nothing'
[org/gnome/desktop/lockdown]
disable-lock-screen=true
DCONF
dconf update || warn "dconf update non-zero"

# ----------------------------------------------------------------------------
# register phase 2 systemd oneshot
# ----------------------------------------------------------------------------
msg "Registering kiosk-postinstall2.service"
UNIT=/etc/systemd/system/kiosk-postinstall2.service
WRAP=/usr/local/sbin/kiosk-phase2.sh
install -d -m0755 /usr/local/sbin /var/lib/kiosk
cat >"$WRAP" <<'WRAP'
#!/usr/bin/env bash
set -e
LOG=/var/log/kiosk_postinstall_phase2.log
exec > >(tee -a "$LOG") 2>&1
echo "$(date +%F\ %T) phase2 start"
[ -f /opt/usb-seed/postinstall2.sh ] && bash /opt/usb-seed/postinstall2.sh
date > /var/lib/kiosk/postinstall2.done
systemctl disable --now kiosk-postinstall2.service || true
echo "$(date +%F\ %T) phase2 done"
WRAP
chmod 0755 "$WRAP"

cat >"$UNIT" <<'UNIT'
[Unit]
Description=Kiosk Phase 2 (VMware + VM extraction + tweaks)
After=network-online.target systemd-udev-settle.service
Wants=network-online.target
ConditionPathExists=!/var/lib/kiosk/postinstall2.done
[Service]
Type=oneshot
ExecStart=/usr/local/sbin/kiosk-phase2.sh
StandardOutput=journal+console
StandardError=journal+console
TimeoutStartSec=0
[Install]
WantedBy=multi-user.target
UNIT

systemctl --no-reload enable kiosk-postinstall2.service || warn "systemctl enable non-zero"

# ----------------------------------------------------------------------------
# completion stamp
# ----------------------------------------------------------------------------
mkdir -p /var/lib/kiosk
date -Iseconds >/var/lib/kiosk/postinstall1.done
msg "Phase 1 complete; reboot will be invoked by user-data."
sync
udevadm settle --timeout=30 || true
msg "==== kiosk postinstall.sh end (exit 0) ===="
exit 0
```

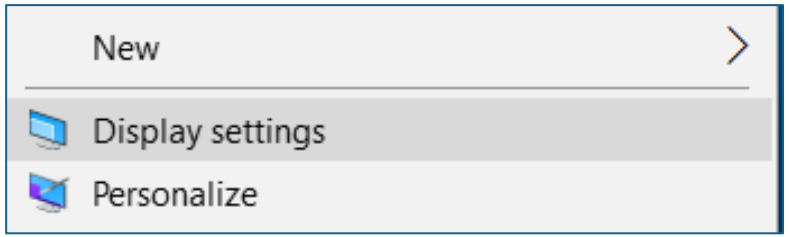
postinstall2.sh (runs on 1st reboot. Install software as 'root' and as 'ubuntu' user

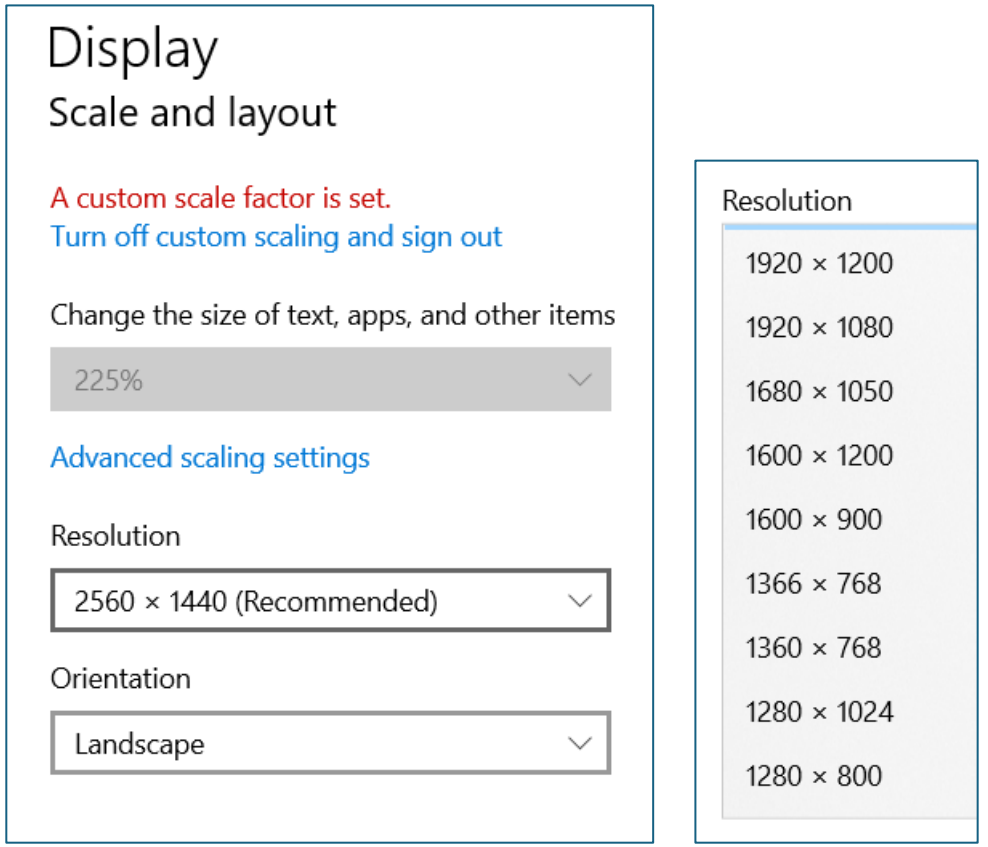
```bash
#!/usr/bin/env bash
# ============================================================================
# postinstall2.sh — Kiosk Post-Install (Phase 2)
# Version: 2.2.2025-10-29
# Order of operations (after desktop loads):
#   1) Close Welcome/Tour windows
#   2) Open terminal: tail -f kiosk_postinstall_phase2.log
#   3) Open terminal: top for VMware processes
#   4) Update sudoers NOPASSWD
#   5) Install VMware Workstation Pro (best-effort)
#   6) Extract VM from 7z
#   7) Flatten nested win10/
#   8) Locate .vmx
#   9) Patch .vmx
#  10) Create autostart helper + desktop entries
#  11) Trust desktop entries (gio) in user session
#  12) GNOME lock/idle settings
#  13) Disable Phase 2 service
#  14) Print reboot banner
# ============================================================================
set -Eeuo pipefail

# ---------------------------
# Logging / Tracing
# ---------------------------
LOG=${LOG:-/var/log/kiosk_postinstall_phase2.log}
mkdir -p "$(dirname "$LOG")"
exec > >(tee -a "$LOG") 2>&1
_ts() { date +"%F %T"; }
log() { printf '[%s] %s\n' "$(_ts)" "$*"; }
hr() { printf '\n[%s] ============================================================\n' "$(_ts)"; }
PS4='+ $(date "+%F %T") [${BASH_SOURCE##*/}:${LINENO}] '
# Auto-enable DEBUG
#DEBUG=${DEBUG:-1}
DEBUG=${DEBUG:-0}
[[ "$DEBUG" == 1 ]] && set -x || true
trap 'rc=$?; log "[ERROR] rc=$rc at line $LINENO: $BASH_COMMAND"; exit $rc' ERR

hr; log "==== Phase 2 start (v2.2) ===="

# ---------------------------
# Configuration (override via env)
# ---------------------------
SEED_DST=${SEED_DST:-/opt/usb-seed}
USER_NAME=${USER_NAME:-ubuntu}
USER_HOME=$(getent passwd "$USER_NAME" | cut -d: -f6 || true); : "${USER_HOME:="/home/${USER_NAME}"}"
USER_UID=$(id -u "$USER_NAME")

VM_ROOT=${VM_ROOT:-"${USER_HOME}/vms/win10"}
VMX_PATH_DEFAULT=${VMX_PATH_DEFAULT:-"${VM_ROOT}/win10.vmx"}
SPLIT_DIR=${SPLIT_DIR:-"${SEED_DST}/split"}
VM_ARCHIVE=${VM_ARCHIVE:-"${SPLIT_DIR}/win10.7z"}
VMWARE_BUNDLE_PATH=${VMWARE_BUNDLE_PATH:-"${SEED_DST}/VMware-Workstation-Full-25H2-24995812.x86_64.bundle"}
SEVEN_Z=${SEVEN_Z:-/usr/bin/7z}
VMRUN_BIN=${VMRUN_BIN:-/usr/bin/vmrun}
VMWPRO_BIN=${VMWPRO_BIN:-/usr/bin/vmware}
VMPLAYER_BIN=${VMPLAYER_BIN:-/usr/bin/vmplayer}
SYS_AUTOSTART_DIR=${SYS_AUTOSTART_DIR:-/etc/xdg/autostart}
USR_AUTOSTART_DIR="${USER_HOME}/.config/autostart"
USR_APPS_DIR="${USER_HOME}/.local/share/applications"
USR_DESKTOP_DIR="${USER_HOME}/Desktop"

```

This file will likely be changed the most to fit your needs and customizations.

Example:
Variables have been defined early in the script to allow changes for file names and folders, e.g. 7zip file names for VMware images.

Resize MS Windows Screen





Resize as needed, depending on the monitors being used.

- Suggest 100% or 125% as needed

Monitoring Commands

Phase 1 - Initial Boot - Operating System Install

sudo tail -f /var/log/installer/subiquity-server-debug.log

top -c

sudo tail -f /target/var/log/kiosk_postinstall_phase1.log

Phase 2 - 1st Reboot - Software Install

tail -f /var/log/kiosk_postinstall_phase2.log

top -c

cat /var/log/kiosk_postinstall_phase1.log

Purpose	Log file	Notes
Main installer log	`/var/log/installer/subiquity-debug.log`	Shows all actions and Python tracebacks.
Installation progress	`/var/log/installer/subiquity-server-debug.log`	Real-time output during partitioning, package install, user creation, etc.
Autoinstall (cloud-init)	`/var/log/cloud-init-output.log`	Contains autoinstall status and errors after first boot.
Kernel messages	`/var/log/kern.log` or `dmesg -w`	Hardware or driver events.

To follow during install (in a shell, like Ctrl+Alt+F2):

bash Copy code

```
tail -f /var/log/installer/subiquity-debug.log
```