HDDSuperClone

This manual is for HDDSuperClone (version 2.3.2 23 January 2022)

Home page: www.hddsuperclone.com

A PDF version of this manual is available in the downloads section

Chapters:

Introduction

HDDSuperClone is an advanced disk cloning tool for Linux. It is used to clone a disk to another disk or image file. It creates a sector-by-sector clone or image of the entire disk. It uses multiple copy phases to achieve the best possible results when there are bad sectors. It uses a progress log file to log the progress of the recovery, and can be stopped and resumed at any time. It has an advanced skipping algorithm that is designed to self adjust to skip out of a bad area or bad head as efficiently as possible. It has many advanced options, but most average users should not need to change those options in most cases.

Important Notes

This is an advanced utility, and it is very possible to send destructive commands to a drive. Please make sure you know what you are doing before using this software. I am in no way responsible for any data loss or damage caused by using this software. Use it at your own risk!

While I do my best to ensure that this software does what it is supposed to without flaw, there is always the possibility of a programming error on my part. If an error is found, I will fix it as soon as possible. However, I assume no responsibility for any data loss or damage caused by any programming error. You are responsible for testing any commands and scripts before using them on critical data.

HARD DISK DRIVE PHYSICAL DAMAGE DISCLAIMER:

When dealing with any hard disk drive, it is always possible that there is physical damage inside. This is even more likely if the drive was dropped. When attempting to read a disk that is damaged, further internal damage can occur. If a head is physically damaged, it can destroy the surface of the platter, which destroys all data on that surface. No software can detect this. It can only be found by examining the drive in a HEPA 100 clean room or laminar flow workbench. NEVER open a drive yourself unless you have a HEPA 100 clean room or laminar flow workbench, and have experience in doing so. If you open a drive outside of those conditions, you will introduce the contamination of microscopic dust particles, which will kill the drive heads and cause damage to the platter surfaces, destroying all hope of recovering the data. So be aware that when dealing with a failing hard drive without knowing what you are doing, you could be causing further damage without even knowing it.

BUFFER / CLUSTER SIZE LIMITATIONS:

There is a maximum program buffer size limit of 4194304 (4MB) for direct IDE and AHCI modes. To get the maximum cluster size for direct IDE and AHCI, you must start the Virtual Disk Driver. Note that when using passthrough mode that there is normally a buffer limit of 524288 (512KB). There is a limit stored at /sys/block/DEVICE/queue/max_sectors_kb, where "DEVICE" is the device you are reading (example "/sys/block/sda/queue/max_sectors_kb"). The number stored here is referenced in KB, and the default for a hard drive is usually 512 (meaning 512KB). This number is usually smaller for a USB connected drive. If you attempt to set a cluster size that would put the buffer above the limit, the cluster size will be reduced to the allowed limit. When using HDDSuperTool, an attempt to set the buffer size greater than allowed will result in an I/O error.

SPECIAL NOTE FOR NTFS DESTINATION IMAGE FILE SPEED:

When the destination is an image file on a NTFS partition, it has been noticed that the write speed can be slower than normal. This has something to do with ntfs-3g package in Linux. In testing it has been found that going into the advanced settings of HDDSuperClone and disabling

the operating system write buffer brings the speed back to normal. This is the opposite of how it should normally work, but for NTFS this seems to be the solution for the write speed issue.

SPECIAL NOTE FOR SAS SCSI DRIVE CLUSTER SIZE:

When working with SAS SCSI drives in passthrough mode, the usable buffer seems to be reduced by over half of the limit. For example, when working with a 520 sector size drive the hard limit for cluster size was 1008 (524288 / 520), but it was likely to fail with some an I/O Error Invalid Argument if the cluster size was set above 504. The issue also seems to affect drives with the normal 512 byte sector size. So if you set the cluster size high and end up with an **I/O Error Invalid Argument**, set the cluster size to half of what the maximum would be.

SPECIAL NOTE FOR ENABLING SCSI WRITES TO DESTINATION:

Enabling the SCSI writes to the destination is ONLY meant for writing to odd sized SAS drives (520 bytes per sector). Never use it for any other purpose. When enabling this feature it has been found that no sense data is returned by the OS, so write error detection is limited. A two second write timeout has been implemented to catch bad writes that could possibly get past the write error detection. If you use this option, it is recommended to read back the data and verify it against the original, if possible.

SPECIAL NOTE FOR USB ATTACHED DRIVES:

USB attached drives are not always predictable. This is dependent on the USB bridge/adapter. The USB bridge/adapter may not send the proper response when there are errors or faults on the device. This could mean that the program will continue when it should exit, and all reads will be bad (or worse, all reads could be considered good when there is not actually data being read!). It is possible that it will not report any errors at all even for bad sectors, making it look like all the data is being recovered when it is not. It may also be possible that it may send a response indicating a fault when there isn't one, causing the program to exit. While I have done my best to properly detect the condition of the device, there is nothing I can do about adapters that respond in odd ways. If possible, it is always recommended to attach the drive directly to the computer via IDE/SATA connection. Use USB adapters at your own risk!

SPECIAL NOTE FOR USB DRIVES WITH HDDSUPERTOOL:

When in normal passthrough mode, it may be possible to send ATA commands to USB attached ATA drives. However, what commands are supported and how the drive reacts to certain commands is entirely up to the USB adapter interface that is attached to (or built into) the drive. This means that some commands may not be supported by the drive, and some commands may give odd results. This is not the fault of HDDSuperTool. For instance, there are some scripts for Western Digital that use vendor specific commands to read modules. These scripts will not work on all USB drives! There is nothing I can do about that. If other commands seem to work fine, but the vendor specific commands do not then the USB adapter interface (or the drive) likely does not support those commands (the commands may be locked out by the manufacturer).

SPECIAL NOTE FOR ATA-PASSTHROUGH ON LINUX KERNEL ABOVE 4.4:

On Linux kernels above 4.4 there is no longer any ATA return data in ata-passthrough mode. Ubuntu 16.04.1 (kernel 4.4) works, but Ubuntu 16.04.2 (kernel 4.8) only returns SCSI sense data. This does not have a major effect on program operation for HDDSuperClone, although certain optimizations will not happen. But this can effect how some of the scripts work in HDDSuperTool.

SPECIAL NOTE FOR NVME SSD DRIVES:

NVME SSD drives will not show up with the default passthrough mode. You must use **Generic source device** mode.

SPECIAL NOTE FOR USING VIRTUAL MACHINES:

HDDSuperClone is NOT meant to be run from within a virtual machine. It should always be run directly on the hardware. While some modes may work within a virtual machine, it is not in any way supported. Also, the direct modes of the pro version require direct hardware access, so they will not work within a virtual machine, and will likely cause a lockup if used within a virtual machine.

Installing

Import information about program dependencies: If you successfully install HDDSuperClone, but get errors when you try to run it, it is possible that some program dependencies are missing. In this case, try the following terminal command to install the dependencies.

For Debian based (Ubuntu):

```
sudo hddsuperclone-install-depends-deb.sh
```

For Red Hat based (Fedora):

```
sudo hddsuperclone-install-depends-red.sh
```

The script will attempt to perform the following actions for Debian:

apt-get install libusb-0.1 -y

apt-get install make -y

apt-get install gcc -y

And these actions for Red Hat:

yum install libusb-0.1 -y

yum install make -y

yum install gcc -y

If successful, there should be no further errors when starting the program. Libusb is important for both free and pro versions, make and gcc are only important for the pro version.

HDDSuperClone is supplied as an executable, with separate ones for 32bit and 64bit. You CANNOT run the 32bit on a 64 bit system and vice versa (you may get an odd file not found error if you try).

HDDSuperClone now has DEB (Debian/Ubuntu) and RPM (RedHat/Fedora) installers available. If you have downloaded the appropriate installer then you should just be able to double click on it to start the installation process. This method should even work on many live CDs, but not all. If you are unable to use one of the installers, then you will need to follow the instructions below to install from the tar.gz file or to run without installing.

If you have downloaded the tar.gz file, then the easiest way to get started is to copy the tar.gz file to a flash drive, and then use simple copy and paste to put it in the Home folder that can be found on the desktop of the Linux install. When you open a terminal it should default to the same Home folder that is on the desktop.

To extract hddsuperclone, open a terminal and use the following commands (replacing the -x.x-x.x.-xxx with proper version number and architecture):
```
gunzip hddsuperclone-x.x-x.x.-xxx.tar.gz
tar xf hddsuperclone-x.x-x.x.-xxx.tar
```

Then navigate to the proper directory:
```
cd hddsuperclone-x.x-x.x.-xxx
```

The following method to install HDDSuperClone will not only work on a Linux installation, but you can use the same method when booting from a live CD. The only difference is that every time you use a live CD, you will need to perform these steps after each boot.

To install hddsuperclone, use the following command:
```
sudo make install
```

The "make install" command needs to be run as root, which is why "sudo" is included in this example. Your sysem may use a different command, or you may already be root. If it is not ran as root, then you will likely get permission errors and the install will not be complete. Note that you can also uninstall it with the command "sudo make uninstall". There is now also an uninstaller script that can be ran by typing "sudo hddsuperclone-uninstall.sh".

You must run hddsuperclone as root. For Ubuntu you would use "sudo" in front of the command:
```
sudo hddsuperclone
```

Please consult the documentation for your version of Linux for information on how to run as root.

To run it without installing, you must be in the same directory as hddsuperclone. Note that some versions of Linux will not allow you to run a program from certain external devices (such as a FAT32 USB drive). Example to run it from the current directory:
```
sudo ./hddsuperclone
```

You may need to change the permissions on the file so that you have the rights to run it. The following command should do that:

```
sudo chmod a+x hddsuperclone
```

If you are booted from a live CD that does not allow installing with make (maybe make does not exist) and you are trying to run it from a USB drive and are getting a permission error, you can copy the executable to the home folder and run it from there. Note that if using a live CD the home folder exists in ram and will be cleared on a reboot. The following example assumes you are in the folder on the USB drive that contains hddsuperclone. The first command copies it to the home folder, the second command gives permission to execute, and the third command runs it:

```
sudo cp hddsuperclone /home
sudo chmod a+x /home/hddsuperclone
sudo /home/hddsuperclone
```

Getting started

To use HDDSuperClone to clone a disk drive, first open HDDSuperClone, either continue in English or change the language at the first popup box and **Continue**.

Once at the main window, choose **New Project** from the file menu. Choose a name and location for the project log file. ***Remember that if using a Live CD, the project log file is stored in RAM and will be gone after a reboot unless you choose a different location, such as a flash drive!***

Choose the source drive and the destination drive or image from the **Drives** menu.

Click the **Connect** button to connect to the source and destination.

To start cloning, click the **Start** button. To stop cloning at any time, click the **Stop** button.

Click the **Disconnect** button to disconnect the source and destination (to change some settings you must be disconnected).

HDDSuperClone can be stopped and restarted as needed. To resume after closing the program, follow the steps above except choose to **Open Project** instead of New Project. Please see the Usage section of this manual for more advanced explanations and settings.

Theory of Operation

HDDSuperClone uses multiple copy phases to achieve the best possible results. To stop the program at any time, click "Stop" in the GUI. The program can also be stopped and closed by pressing ctrl-c from the terminal window. A progress log file is used to keep track of recovery progress, and when resumed will pick up where it left off, unless you reset the status, or in some cases if you change the options you may alter how it resumes.

Phase 1 is a copy pass forward with skipping. **Phase 2** is a copy pass backward with skipping. Together they offer the best attempt to get the most good data first and fast from the good areas of the drive. The skipping is based on read errors and will skip when a read error is encountered, and self adjusts to try to skip out of a bad head in about 7 read errors. A read that takes longer then the –skip-threshold value will also trigger a skip event. The skipping starts out at –skip-size, and increases as needed per the algorithm, unless it hits –max-skip-size in which case it is reset back to –skip-size and the program will stop with an error message. These two passes are the money passes. If after these two passes you don't have a percentage complete in the upper 90's, then you likely have a weak/damaged head, and the next phases could take a long time to complete.

Phase 3 is a copy pass forward with skipping. But the skipping size for this pass does not self adjust, and skipping is based on the read rate instead of read errors. If the read rate is below the value of –rate-skip for two consecutive reads, it skips ahead the amount of –skip-size.

Phase 4 is a copy pass forward without skipping. This gets all the remaining non-tried areas. All failed blocks larger than one sector (LBA) in size are marked as non-trimmed by the first 4 phases. Failed blocks that are one sector in size may be marked as bad if timeouts are not used.

Trimming reads each non-trimmed one sector at a time forward until a read error, and then backward until a read error. Any trimmed blocks larger than one sector are marked as non-scraped.

Dividing is only performed if trimming is turned off. If trimming is turned off, then 1 or 2 dividing copy passes are done instead. The default is for only one dividing pass, to activate the second pass use the –do-divide2 option. If there is only one dividing pass, then it reads the non-trimmed blocks with the cluster size / 8, and marks the failed blocks as non-scraped. If the –do-divide2 option is used, then the first dividing pass reads non-trimmed blocks with the cluster size / 4. The second dividing pass reads non-divided blocks with the cluster size / 16. The first dividing pass marks failed blocks larger than one sector as non-divided, the second pass marks them as non-scraped. Trimming has been found to be more efficient than dividing with scsi-passthrough mode. Dividing can be more efficient with ata-passthrough mode (not in the free version as the marking of reported bad sectors is disabled) and the direct modes (more so with direct mode when using timers), but how efficient depends on how the drive is reacting.

Scraping reads the non-scraped blocks one sectors one at a time forwards. Failed sectors are marked as bad.

Retrying retries all the bad sectors one at a time forwards.

Usage

HDDSuperClone must be run as root. The format for running hddsuperclone is:

`hddsuperclone [`*`options`*`]`

HDDSuperClone supports the following options:

`'-h'`
`'--help'`

    Print an informative help message describing the options and exit.

`'-v'`
`'--version'`

    Print the version number and exit.

`'--tool'`

    Start HDDSuperTool. See the [HDDSuperTool Usage](#) section for more information.

Language Selection

When HDDSuperClone is first started in GUI mode, there will be a box with two buttons. The top button is to Continue to open HDDSuperClone, the bottom button is to change the language. When you click on Change Language, a file menu will open to a location where the language files are located. Choose the one that matches your desired language and then click Open. You should see a popup message that the language was changed successfully, click OK and then Continue.

The language files are created using Google Translate. Because of this, there will possibly, if not likely, be some translation issues. However, I believe that it should be good enough to be able to operate the program.

Update: I am now adding additional translation versions of language files that have been provided by others. They will have a dash "-" after the language name, and either a number or some other identifying method, depending on information provided by the author. If for some reason one of these additional language files does not work, it would likely be my fault for not getting it updated to match any updates I make in the program.

If you wish to submit a language translation pack, it must be in the format that HDDSuperClone can open. And hopefully you are willing to help with updates as changes and additions to the program are made.

The main window of HDDSuperClone has the following buttons on the side:

Connect – Connects the program to the source and destination devices.
Disconnect – Disconnects the program from the source and destination devices.

Start – Starts the selected operation.

Stop – Stops the current operation.

Analyze – Starts the process of analyzing the device.

Extended Analyze – Extended version of analyzing the device.

SMART – Gets and displays the SMART status of the device.

Clone Mode – Selects the operation of cloning.

Slow Driver – Selects whether or not to use the slow driver mode.

Virtual Mode 1 – Selects the virtual disk device driver operation mode 1.

Mode 2 – Selects the virtual disk device driver operation mode 2.

Mode 3 – Selects the virtual disk device driver operation mode 3.

Mode 4 – Selects the virtual disk device driver operation mode 4.

Mode 5 – Selects the virtual disk device driver operation mode 5.

Soft Reset – Performs a soft reset on the source device in direct mode. In other modes it reconnects to the device.

Hard Reset – Performs a hard reset on the source device in direct mode. In other modes it reconnects to the device.

Power Off– Activates the primary relay.

Power On – Deactivates the primary relay.

The main window of HDDSuperClone has the following menu items at the top:

File-->Open Project – This opens a new project file (also called a progress log file). This is the progress log file used for resuming the recovery.

File-->New Project – This opens an existing project file.

File-->Import ddrescue log (map) file – This will import a ddrescue style map file. You must choose New Project first to be able to import a ddrescue log file into the project.

File-->Save Project – Saves the current project. Note that the project is automatically saved during the cloning process every 30 seconds, and at other times when making settings changes.

File--> Save Project As - Saves the current project as an alternative file name. This does not change the current project name.

File-->Export ddrescue log (map) file – Exports the current project as a ddrescue style log file. This does not change the current project.

File-->Add to Domain from DMDE bytes file – Use a cluster list or file list bytes file from DMDE to add the targeted data to the domain for the purpose of targeted cloning (not available in free version). The cluster list (Cluster Map Report) must be in the form of "in bytes from the disk start". The file list (List sectors to file) must be in the form of "bytes". Failure to use the format of bytes will result in incorrect data targeting.

File-->Load domain file – Use a domain file to limit the recovery area to a domain. The domain file can be in the same format as the standard progress log file, or a ddrescue log file, the format

will be auto detected. Areas marked as finished in the domain will be recovered. Areas marked as anything else will not be tried. The domain can be loose, meaning that there can be gaps and the start and end positions are not limited.

File-->New domain file – Start a new domain file to use with the driver mode.

File-->Clear Domain – Clears any loaded domain data from memory**.**

File-->Save domain file – Saves the current domain file**.**

File-->Save domain file as – Saves the current domain file with a different file name**.**

File-->Quit – Quit HDDSuperClone and exit.

Mode-->Generic source file – The source will be a file, when choosing the source a file dialog will open.

Mode-->Generic source device – The source will be a block device, when choosing the source it will list available block devices. You should use this mode if the device does not show up for the default passthrough mode, such as for NVME devices.

Mode-->Passthrough auto detect – Automatically chooses ATA passthrough if capable, otherwise uses SCSI passthrough.

Mode-->Passthrough SCSI – Use SCSI passthrough mode. This is the mode that can work with a USB attached device.

Mode-->Passthrough ATA – Use ATA passthrough mode. This can have a benefit of bypassing some of the kernel driver retries that may be performed in scsi mode. While this may list a USB attached drive, a USB attached drive cannot be cloned in ata mode.

Mode-->Direct IDE – Use direct IDE mode (limited in free version). The drive must be hidden form the OS to use this mode. The drive must be either a PATA connected drive, or the BIOS must be set to IDE mode for a SATA connected drive.

Mode-->Direct AHCI – Use direct AHCI mode (not available in free version). The drive must be hidden form the OS to use this mode. The drive must be a SATA connected drive.

Mode-→Direct USB – Use direct USB mode (not available in free version). This has the benefit of bypassing the OS drivers when accessing a USB attached mass storage device, and gives the possibility of timeout resets for USB devices. Use this mode for devices such as flash drives.

Mode-→Direct USB ATA – Use direct USB ATA mode (not available in free version). This is the same as the regular direct USB mode, except it performs some commands that are meant for conventional ATA drives (including standard SATA AHCI SSD drives). You should not use this mode for flash drives, as the commands can lock up the device, causing a failure when trying to connect.

Mode-->Fill Zero – Fill all non-finished areas of the destination drive or image with zeros. Any area that is not marked as FINISHED will be filled. You must supply the progress log file, but the source is not required for this command. This command will ignore any domain file (the domain file will have no effect). The The settings Input offset, Size, and Output offset do affect this command, as it will only fill areas that are within the limits set by those options if they are set.

Mode-->Fill Mark – Fill all non-finished areas of the destination drive or image with a marking pattern, and then exit. Any area that is not marked as FINISHED will be filled. You must supply

the progress log file, but the source is not required for this command. This command will ignore any domain file (the domain file will have no effect). The settings Input offset, Size, and Output offset do affect this command, as it will only fill areas that are within the limits set by those options if they are set. The format for the fill pattern is as follows: At the start of every LBA will be the text "HDDSUPERFILLMARK", followed by a space, and then the status of the block according the progress log (NON-TRIED, NON-TRIMMED, NON-SCRAPED, NON-DIVIDED, BAD). There will be another space and then the number of the LBA as text proceeded by "0x" to indicate the number is in hex (example 0x1a2b3c). There will be a few more spaces, and then the rest of the LBA block will be filled with a repeating hex pattern of dd ee aa dd bb ee ee ff.

Mode-→Erase Destination – Use this mode to erase a destination disk before cloning to it. You need to start a new project to begin erasing, or open an existing erase project to resume erasing. If you have issues with it locking up, you may wish to disable the Operating System write buffer in the advanced settings. Erasing is just writing zeros to the disk, and the OS buffer can get way ahead of the actual writes to the disk, which can cause issues when trying to choose a disk in another instance of HDDSuperClone. However, disabling the write buffer can make it slower. You must be disconnected for a change to the write buffer setting to take effect.

Mode-->Virtual Driver only – This mode does not read any data from a source. It is for use with the driver mode where there has already been data recovered, and no data is needed or desired to be read from the original source. Data is only read from the destination.

Drives-->Choose Source Drive – Choose the source drive to recover.

Drives-->Choose Destination-->Generic block device – Choose a block device as the destination.

Drives-->Choose Destination-->Drive – Choose the destination drive.

Drives-->Choose Destination-->Image file – Choose the destination image file.

Drives-->Choose Destination-->NULL (No destination) – Choose not to use a destination, no data will be recovered.

Note that when choosing a drive, the number in parentheses is the size of the drive in bytes.

Drives-->Start Virtual Disk Driver – Starts the virtual disk driver so that the driver mode can be used. May also be needed for DMA memory mapping if the program is unable to get memory in the 32 bit range by the normal method.

Drives-->Stop Virtual Disk Driver – Stops the virtual disk driver.

Drives-->Fix Driver Memory Error – Fixes the issue when the virtual driver will not load with a memory allocation error (due to not enough contiguous memory).

Drives-->Disable Ports – Disable ports for the purpose of hiding drives from the operating system.

Drives-->Restore original startup port configuration – Restore the original GRUB configuration to clear any changes from disabling ports.

Drives-->Disable USB mass storage driver – Disable the USB mass storage driver so that the OS will not try to access any USB attached drive.

Drives-->Restore USB mass storage driver – Restore the USB mass storage driver.

Settings-->Clone Settings – Open the Clone Settings dialog

Settings-->Advanced Settings – Open the Advanced Settings dialog

Settings-->Timer Settings – Open the Timer Settings dialog

Settings-->Primary Relay Settings – Open the Primary Relay Settings dialog

Settings-->Choose Primary Relay Board – This will open a window to select available relay boards. Only recognized relay boards will be listed.

Tools-->Reset Current Status – Resets the current status and position back to the default starting point of non-tried. It also resets all skipping data. This can be helpful if your settings cause runaway skipping and there have been skip reset events.

Tools-->Repair Log – This will attempt to repair the progress log file in the event that it is incomplete, damaged, or otherwise corrupt, and then exit. A source drive must be selected as the log end is compared to the size of the drive. Any repaired areas where the status cannot be known for sure are marked as non-tried. You should not normally need to use this option.

Tools-->Reset Log – This will reset the position to 0, the current status to non-tried, and will change all unfinished blocks in the log to a status of non-tried. This will also reset all skipping data. This option is useful if you need to make major changes to the settings and need to start over, but don't want loose the status of any finished blocks.

Tools-->Display Identify Results – This will display the identify device results.

Tools-->Display Analyze Results – This will re-display the past results from analyzing the device.

Tools-->Display SMART Results – This will re-display the past SMART results from the device.

Clone Settings

Phase 1 – Enable phase 1.

Phase 2 – Enable phase 2.

Phase 3 – Enable phase 3.

Phase 4 – Enable phase 4.

Trim (overrides dividing) – Enable trimming phase.

Divide – Enable dividing phase.

Divide 2 – Perform two dividing phases instead of just one if Divide is enabled and Trim is disabled.

Scrape – Enable the scraping phase.

Reverse – All operations are done in the opposite direction. This means that most of the operations are done in reverse, but since phase 2 is normally in reverse, when this option is used phase 2 is a forward pass. If you only want to read in reverse then also disable phase2. While

trimming it done opposite, it still reads in both directions, so if you REALLY only want to read in reverse only then also disable trimming.

Skip Fast – Not available in free version. Enables a more aggressive skipping algorithm to skip out of a bad head faster. This does increase the possibility of over skipping, especially on a single platter two head drive.

Mark Bad Head – Not available in free version. Uses the skipping information during phases 1 and 2 to mark what is considered to be a bad head in the log.

Read Bad Head – Not available in free version. If unchecked, any areas that were marked as bad head in phases 1 and 2 (or Rebuild Assist phase) will not be read in further phases.

Rebuild Assist – Not available in free version. Enables the Rebuild Assist phase, if Rebuild Assist is enabled in the Advanced settings. If it is not enabled in the Advanced settings, then this setting has no effect. This phase takes the place of phase 1. After a successful completion of this phase, the setting will disable itself so the Rebuild Assist phase will not run again in future passes.

Retries – The number of retry passes to perform.

Cluster size (LBA) – Number of sectors (LBA) per read block in copy phases 1-4, default 128.

Sector size (bytes) – The logical sector size in bytes. The program attempts to get this information automatically when choosing the source drive.

Input Offset (LBA) – LBA offet to start reading from, default 0.

Size (LBA) – LBA to read starting from Input Offset, default is until the end of the drive.

Skip Size (LBA) phase-1-2-3 – Starting skip size in LBA, the default is 4096 LBA. This option only applies to copy phases 1, 2, and 3. In copy phases 1 and 2 this number can grow to adjust to the head skipping algorithm. In copy phase 3 this number stays fixed.

Max Skip Size (LBA) phase-1-2 – Maximum skip size in LBA. The default is calculated as drive size divided by 1000, or 1 GB, which ever is smaller. This option only applies to copy phases 1 and 2. The actual current skipping size is limited to this value. You should normally not need to adjust this.

Skip Threshold (ms) phase-1-2 – The time in milliseconds for a read that will trigger a skip event, the default is 1000 (1 second). This option only applies to copy phases 1 and 2. This is meant for drives that have very slow reading areas, but not enough if any errors in those areas to trigger the skipping algorithm. Any read that takes longer than the value of the setting will trigger a skip event. Note that for drives that have a "slow responding" issue this will cause runaway skipping and you will see the skip reset count increase. If this happens you may need to use the –reset-status option and start over with adjusted options. To disable this threshold just set it to a high number such as 60000 (60 seconds).

Rate Skip (kB/s) phase-3 – Rate for rate skipping pass in kilobytes per second, the default is 50 kilobytes per second. This option only applies to copy phase 3. If the rate is below this setting for two consecutive rate updates, then the position skips ahead by the starting skip size.

Exit on Slow (kB/s) phase-1-2 – Rate in kilobytes per second for exiting on slow reads, the default is -1 (disabled). This option only applies for copy phases 1 and 2. If the –exit-slow-time has not been set, and the read rate is below this setting for two consecutive rate updates, then the program will stop If the –exit-slow-time has been set, and the read rate is below this setting for longer than the slow timer value, the program will stop. This option also increases the skip-threshold to a very high setting so that it will not induce skipping, unless the skip-threshold

setting is set manually. This option is meant for drives that suffer from a slow responding issue, where power cycling the drive when it becomes slow will seem to bring it back up to speed for a short time.

Exit on Slow Time (sec) phase1-2 – This is an optional timer setting in seconds to complement the –exit-on-slow option, the default is -1 (disabled). If set, this is the number of seconds of slow reading before the program will stop.

Block size (LBA) – Not available in free version. The number of logical sectors per physical sector. All reads will be aligned with this value, and this will be the smallest read size. This is useful for advanced format drives, to make the reading more efficient. The program attempts to get this information automatically when choosing the source drive. Not all drives may return this information.

Alignment offset (LBA) – Not available in free version. If there are multiple logical sectors per physical sector, and the logical sectors have been offset (to accommodate for older operating systems such as XP), this value should be set. It works with Block size to align the reads for optimum performance. The program attempts to get this information automatically when choosing the source drive. Not all drives may return this information.

Max read rate (kB/s) – Not available in free version. Limit the maximum read speed in kilobytes per second. The default value of 0 is unlimited. This can help stabilize drives that get too hot when reading fast, such as SSD and NVMe devices.

Advanced Settings

Do not create backup log – By default every time the progress log file is written, the current file is written with a ".tmp" extension, then the last log is renamed with a ".bak" extension, and then the tmp file is renamed to the log file. This is to help prevent the possible loss of the progress log file due to unforeseen circumstances. Using this option will turn off the creation of the backup file. It does not turn off the use of the tmp file. If you end up with a log file ending with the .tmp extension, that file is likely incomplete and corrupt, and the regular or backup file should be used to continue.

Enable phase logs – When enabled a log with a time and phase stamp in the name will be created at the end of every phase. This can be useful for examining the progress from each phase.

Force access of mounted disk – Force working with the source even if it is detected to be mounted. It is not recommended to attempt to recover a drive if it is mounted.

Force same controller / slave – Force working with the source even if it is detected that there is another drive on the same controller, or the drive is a slave on the controller. This if for direct IDE mode only. It is not recommended to use this option, however there may be times when a drive and controller are locked up and report these conditions when they are not true. This is a method to get past that condition.

Enable changing the output offset – Enable the ability to change the output offset. You must close and reopen the Advanced Settings for this change to take effect.

Output Offset – LBA offset of destination writes, the default value is -1 (follow input offset). This is considered an advanced option. Under normal circumstances you should not use this option. Do not use this option unless you know what you are doing! Using this option improperly could cause total corruption of data on the destination!

Enable changing current position – Enable the ability to change the current position. You must close and reopen the Advanced Settings for this change to take effect.

Current position– LBA current position. It is not recommended to change this until after phase 1 and 2 are complete, because it will affect the skipping algorithm. Use this at your own risk, jumping the position around without knowing what you are doing could cause some data to not be recovered.

Action to perform on major drive error – A major drive error is something other than a typical read error. The Exit on Slow condition is also considered a major error.

– Stop cloning with error message – Stop cloning and show an error message indicating the fault.

– Activate primary relay – Activate the primary relay board, which is chosen under the primary relay settings.

– Call command – Call the command specified, and then continue cloning. If the command returns a non-zero value, then cloning will stop with an error message.

– Test command – Test the specified command.

Operating System write buffer – Enable or disable the OS write buffer for writes to the destination. Writes are usually faster if this is enabled. Disabling this is safer if there is the possibility of a power outage. If the write buffer is enabled and the system crashes, there could be unwritten data that would be indicated as recovered in the log.

Virtual Disk Device Driver output options – Options for how the virtual disk handles read errors as presented by the recovery.

– Return read error – Read errors will return with a standard I/O error.

– Return zero filled data – Read errors will return with zero filled data.

– Return mark filled data– Read errors will return with mark filled data. The pattern for mark filled data is repeating 16 bytes of text and 16 bytes of hexadecimal, HDDSUPERFILLMARK and 0xddeeaaddbbeeeeffddeeaaddbbeeeeff.

Virtual Disk IO SCSI only – Only use IO SCSI for reading the virtual disk. Normal block reads will return all zeros when this is enabled. This is used with DMDE IO SCSI to improve stability and help prevent lockups.

Virtual Disk device name – The name of the virtual disk in /dev. The default name is sdv (/dev/sdv). Note that if you choose a name that is already in use, it will cause a system crash. Also, you should choose a conventional name with the SDx format, or some software will not find or list it.

Virtual Disk minimum cluster size – This is for virtual disk mode. When in mode 3, this will be the minimum cluster read size. When cloning using a domain, this will also be the minimum cluster read size, so that small domain reads can be prevented if desired.

Use physical sector size as logical for virtual disk – This will use the physical sector size as the logical sector size when presenting the virtual disk to the operating system. The physical size is determined by the **Block size** setting in the Clone Settings.

Do not perform identify when listing source devices – This will disable the identify device command when listing source devices in Direct IDE and Direct AHCI modes. This is to help prevent a system lockup or error when listing possible source drives. If any disks are performing operations, the identify command can cause an issue. If you are attempting to run multiple instances at the same time of HDDSuperClone in direct IDE or AHCI mode, you should enable

this option to prevent one instance causing an error in another instance. When this mode is enabled, there will not be any model or serial number present when listing drives during choosing a source device. After choosing the device, it will perform the identify command to gather data about only that device.

PIO mode – Use PIO instead of DMA.

Enable using ATA return to mark bad sectors – Not available in free version. This only works with ATA passthrough and Direct modes. When a cluster read is bad, the ATA return data can include the LBA of the first reported bad sector for the read. When this option is enabled, the program will use this data to attempt to read the indicated good data up to the bad sector, and then mark the reported sector as bad and the rest of the chunk as non-trimmed.

Enable read twice on timeout – Not available in free version. This only works with Direct modes that can utilize the soft reset timeout. This option is meant to assist with slow responding drives, where there is a background process that is causing reads to be slow. When enabled and the soft reset timeout is exceeded when reading, after the reset a second read attempt will be performed. The second read attempt will use the Retry reset time as the soft reset time for the this second read attempt. The idea is that sometimes a reset will help prevent the background process for a short time allowing for faster reading, and setting a low soft reset time will cause the reset, but without marking the read as failed.

Retry connecting on failure after relay power cycle – Not available in free version. If the device fails to identify after a relay power cycle, enabling this option will cause a second attempt to disconnect and reconnect the device. It will disconnect, wait for the reset wait time, and then reconnect. This can help in some circumstances.

Enable SCSI write to destination (for odd sector size) – PLEASE SEE THE NOTE IN THE IMPORTANT NOTES SECTION ABOUT USING THIS OPTION! If the destination drive sector size is not a multiple of 512, you must enable this to be able to write to the drive. This would be needed to clone a 520 byte per sector drive to another 520 byte per sector drive. You do not need to enable this to write to an image file, it is only for writing to a drive.

Enable output sector size adjustment – Enable the ability to change the output sector size. You must close and reopen the Advanced Settings for this change to take effect.

Output sector size adjustment – This is used to convert odd sector sizes such as 520 bytes per sector to 512 bytes per sector. A positive value would mean that the output sector size would be greater than the input sector size by the value, a negative value would mean that the output sector size would be less than the input sector size by the value. For example, to clone a 520 byte sector drive to a 512 byte drive/image, the sector size would be 520 and the value of this option would be -8.

Use button colors in bottom status bar – This option will turn on using colors on the bottom status bar. This does not work on some newer systems, which is why it is now an option. The normal mode of the status bar is a simple active / non-active to indicate status. The colors look nicer and stand out more, if your system supports it.

Timer Settings

All of these timer settings are only for Direct IDE and Direct AHCI mode. They are also for Direct USB mode, but they work a bit differently for that mode. They have no effect in

passthough mode. The general timer may, in some rare cases, have some sort of effect in passthrough mode, but this program has no control over what, if any, effect it has.

Initial busy wait time – This is the amount of time in milliseconds to wait for the drive to be ready when waiting to attempt to perform a read. If this timer expires then a reset procedure is initiated, and the read will be attempted after the reset. This timer also affects the wait time per device when listing the choices for the source device, although no reset is performed.

Busy wait time – This is the amount of time in milliseconds to wait for the drive to become ready when performing an identify device command, usually after an error. If this timer expires then a reset procedure is initiated.

Soft reset time – This is the amount of time in milliseconds to wait for a command to complete before initiating a reset procedure. This is the main reset timer. When it expires a soft reset will be initiated.

Hard reset time – (AHCI only) This is the amount of time in milliseconds to wait for the drive to become ready after attempting a soft reset. When it expires, a hard reset is initiated. If this timer is set to 0, the soft reset will be skipped and it will jump strait to a hard reset. For Direct USB, this timer is the amount of wait time for the device to become ready after a reset before the next level of reset is attempted. If the reset is the last on the list, then it would use the Reset timeout instead.

Reset timeout – This is the amount of time in milliseconds to wait for the drive to become ready after a reset attempt (in the case of Direct USB, the last reset in the list). When it expires, the action command will be performed. If this timer is set to 0, the hard reset will be skipped and it will jump strait to the action command. Note that on drives that do not respond to the soft or hard reset in a timely manner, when this timer expires the action will be performed. If you are not using a relay or command, the program will stop with an error message. For this case, you will either need to turn up this timer or the soft reset timer to a higher value to prevent the program from stopping.

General timeout – This is the amount of time in milliseconds before giving up any attempt and returning with a major error. You would normally set this higher then any of the other timers. If you set a timer higher than this general timer, then that timer's action will not be performed.

Action for reset timeout – The action performed when the reset timer expires.

– Do nothing – Do not perform any command, just return with a major error.

– Activate primary relay – Activate the primary relay board, which is chosen under the primary relay settings.

– Call command – Call the command specified, and then check if the drive has become ready. If the drive does not become ready, it will return with a major error. If the command returns a non-zero value, then cloning will stop with an error message.

– Test command – Test the specified command.

Phase timers – Soft reset timer settings for individual phases. When enabled, the soft reset timers can be set for the different phases. When disabled the normal soft reset time is used for all phases.

Always wait for reset timers timers – When enabled in Direct AHCI or IDE modes, this option will always wait for the next reset time (either the hard reset time or the reset timeout) before checking if the drive is ready. Checking if ready still uses the reset time, so if the drive is not ready, it could wait for up to the reset time again, making for twice the delay. This is meant for

drives that seem to be ready after a soft/hard reset, but are not truly ready yet and need a delay before proceeding after the reset.

USB reset choices – The different USB resets available. Note that these are performed in order if enabled.

– Enable USB bulk reset – This is the standard reset for a USB mass storage device. Disabling it here only disables this as the first reset attempt. It will still be performed along with any of the other resets if they are enabled. It is recommend to leave this enabled.

– Enable USB soft reset – This is only for ATA drives, and will attempt to send a passthrough soft reset command to the drive. This command may not be supported by the USB bridge. It is recommend to leave this disabled, as the bulk reset would normally take care of whatever reset is needed.

– Enable USB hard reset – This is only for ATA drives, and will attempt to send a passthrough hard reset command to the drive. This command may not be supported by the USB bridge. It is recommend to leave this disabled, as the bulk reset would normally take care of whatever reset is needed.

– Enable USB port reset – This performs a port reset, to reset the USB communication to the device. It is recommend to leave this enabled. If you find the need to perform power cycles on the device, then you may wish to disable this to save time, as it can double the reset time before the power cycle is performed.

Primary Relay Settings

Not available in free version. These are the settings for a primary relay board for power cycling a disk drive. To use the primary relay board, you must choose it as an action in Advanced Settings and/or Timer Settings. Note that when the relay board is activated, it will display status messages in the console.

For information about how to connect the relay board, visit the following website link:
http://www.hddsuperclone.com/sitev1/hddsuperclone/usbrelay

The following relay boards can be controlled:

1) Generic USB HID red relay boards. To find these relays, search on ebay for "USB Relay Programmable Computer Control For Smart Home". You may also be able to find them on Amazon. More information about these relay boards can be found at the following links:

http://vusb.wikidot.com/project:driver-less-usb-relays-hid-interface

https://github.com/pavel-a/usb-relay-hid

Current relay board – This displays the currently selected USB relay board, if one was chosen.

Relay board settings – These settings control how the relays act. There are settings for up to 8 relays. Any relay value higher than the number of relays the board actually has will have no effect. When the board is activated, the selected relays under **Activation condition** will energize, and relays not selected will be de-energized. When the board is no longer activated (deactivated), the relays chosen under **Deactivation condition** will be energized, and relays not selected will be de-energized.

Activation time (sec) – This is the amount of time in seconds the relays will be energized under the Activation condition.

Post activation delay time (sec) – This is the amount of time in seconds after the relays have been deactivated before attempting to resume the recovery. This is to give the disk drive time to spin back up and become ready before trying to read from it again.

Test relay operation – Clicking this will cycle the relay board, complete with timer settings.

Activate (Power Off) – Manually activate the relay to the power off condition.

Deactivate (Power On) – Manually deactivate the relay to the power on condition.

Choose Primary Relay Board

Not available in free version. This will list all supported relays that are currently connected to the computer, and allow choosing the desired relay.

Understanding the Display

Project is the project log file for the current recovery.

Destination is the destination drive or image file where the data is to be recovered to. This can also be set to "NULL (No destination)", in which case the data is not written anywhere.

Source is the source drive to be recovered.

Total LBA is the reported size of the source device in LBA. **LBA to read** is the selected read size. By default it will be the same as the total size, unless you selected a different size in the options. **Domain size** is the LBA to read as adjusted by the domain. Any data that is not in the domain will stay marked as non-tried in the display.

Current position is the current position in LBA. **Current status** shows the current copy phase operation.

Clone/Driver Mode shows if the mode is the normal Clone mode, or shows which driver mode is currently being used if in Driver mode.

Run time format is days:hours:minutes:seconds. **Remaining time** is the estimated time remaining and has the same format as the run time. This estimate is based on about the last 5 minutes of run time. There is no way to get this totally accurate, it is a calculated best guess. If there are a lot of errors or a large bad area then this does not start to get accurate until it is several minutes into the trimming phase. **Current / Recent / Longest read time** is the highest read time in milliseconds, the first number being the value since the last display update, the second number

is the value within the last 256 display updates, the third number is the value since the start of the operation.

Current rate is the successful read rate since the last display update. **Recent rate** is the average rate of about the last 5 minutes. **Total rate** is the average rate since the operation was started.

Filled is the numer of LBA that have been filled using fill mode, and also when using erase destination mode. **Retried** is the number of LBA that have been retried in the current operation. **Remaining retry passes** is the number of remaining retry passes.

Last read size (takes the place of **Filled** in virtual device driver mode) will contain three numbers. The first number is the small chunk size that was requested, the second number is the large chunk size that was requested, and the third number is the size of the actual device read. This is to be able to see the read sizes when in virtual driver mode to know if a mode change may be needed.

Finished, **Non-tried**, **Non-trimmed**, **Non-divided**, **Non-scraped**, and **Bad** show the number of LBA that exist for each status type, along with the size in GB. It also shows the number of separate areas for each, and the percentage of each based on the LBA to read.

Skips is the total number of times the program has skipped since the program was started. **Skip runs** is how many skip runs have happened since the program was started. If you see the run count growing, it likely means there is a weak/damaged head. **Skip resets** is the number of times the skipping was reset due to exceeding a limit of 64 skips in a run. ***Under normal circumstances the reset count should remain at zero***. If it is greater than zero it is an indication of a problem with either too large of an initial skip size, too small of an initial skip size, a "slow responding" condition, or it is not reading much if any data, and you may need to adjust the skip size and/or skip threshold to handle a special condition. **Base skip size** is the current base skip size. This can grow if the head skipping algorithm is triggered. **Last run size** is the size of the last skip run, which can be useful in estimating the read size of the head. **Slow skips** is the number of skips that were due to slow reads and not errors.

Data preview is a sample of data from the last good read since the last display update. If there was not a successful read since the last display update, then this will be all zeros.

The bottom status bar contains information from the ATA status and error registers. In SCSI passthrough mode it will be grayed out. In ATA passthrough mode it may (if supported by the OS) indicate the last reported status returned before the display update, and is not (and cannot be) a real time display of the registers. When using the Direct modes it shows real time results that are updated about every half second. The left side of the status bar is the status register, and the right side is the error register. The following are brief descriptions of those fields. Please note that some are depreciated and may or may not used by modern drives, and may have different meanings for different commands and versions of the ATA standard. If you want to know more, do an internet search for "ATA status error register" or "ATA Command Set".

Status register:

BSY - (Busy) the device is busy doing something.

DRDY - (Device Ready) the device is ready to accept commands.

DF - (Device Fault) the device is in a fault condition.

DSC - (Device Seek Complete) heads are settled over a track, now deferred write error.

DRQ - (Data Request) indicates that the device is ready to transfer data.

CORR - (Corrected Data) now alignment error.

IDX - (Index) now sense data available.

ERR - (Error) indicates that an error occurred during execution of the previous command.

Error register:

BBK - (Bad Block Detected) now interface CRC error.

UNC - (Uncorrectable Data Error) indicates an uncorrectable data error has been encountered.

MC - (Media Changed) now obsolete.

IDNF - (ID Not Found) indicates the requested sector's ID field could not be found.

MCR - (Media Change Requested) now obsolete.

ABRT- (Aborted Command) indicates the requested command has been aborted.

TKONF - (Track 0 Not Found) now indicates end of media.

AMNF - (Address Mark Not Found) now command timeout, or other possible definitions.

Analyze Results Explained

To analyze a device, you must choose a project, source, and destination. The destination can be NULL if you do not plan on using the data read to continue with recovery of the device. Any data read from the source during analyzing will be copied to the destination just as if you are cloning, and included in the progress log. With the regular analyze, the first stage will last between 15 seconds and 2 minutes, and will be divided into 8 evenly spaced zones spanning the entire device. The second stage is a variance timing test, and will last 15 seconds. With the extended analyze, the first stage will take between 2 minutes and 16 minutes, and be divided into 32 zones. The second stage will last 1 minute. The results are then displayed in a popup box, along with being printed to the console, and included in the progress log file. The reason to perform the shorter regular analyze first is so that if the drive has a bad condition, you can choose not to continue so as not to cause any further possible damage. If it looks good enough to continue, then you should run the extended analyze to get the best results. Note that the extended analyze is not available in the free version.

The analyzing process attempts to perform an even number of reads per zone in levels, up to a predetermined limit. If it detects it would not be able to complete the next level, it will not attempt it. Even so, the results may not always have an even number of reads per zone, depending on how the drive was responding.

The first three lines indicate the percentage of Good, Bad, and Slow reads.

The next three lines are a percentage indication of possible issues. The **Slow Responding Firmware Issue** is the chance that there is an issue that causes good reads to be slow due to a firmware background operation. It is very difficult to know this for sure, so as a general idea if it is below 50% then it is not likely, if it is between 50-100% then it is possible or likely, and if it is over 100% then it is very likely. The **Partial Access Issue** is the chance that there is a problem with reads past a certain point, likely caused by a firmware translator issue. This number can also go above 100% because of how it is calculated, but generally any value is a possible indication of this issue. The **Bad Or Weak Head** is the chance of exactly that. This value does not go over 100%, and is an indication of how widespread the damage is across the drive.

The next section is the variance timing of reading the same sectors over and over many times. The sectors read are determined by the fasted reads for each zone when analyzing the device. The number in parentheses is the total number of reads performed during the variance test. The data is presented in a low/high format. The low value is the lowest time in milliseconds it took to read, and the high value is the highest time it took to read. This is designed to give a possible indication of any sort of slow responding issue. If you see an extreme variance between any of the low and high values, it would be an indication that something caused the read speed to be much different.

The last section gives information about the reads divided into zones. The first line is a total of all the zones, then the following lines are for the individual zones. The total is the total number of read attempts. Good is how many reads were successful. Bad is how many reads failed (the number in parentheses is how many of those failed reads were from a soft reset timeout). Slow is how many reads were slow (based on the Skip Threshold). Low is the time in milliseconds of the fastest read. High is the time of the slowest read. Average is the average time of all the reads.

Skip Resets

During phases 1 and 2, there is an advanced self learning skipping algorithm applied. But the algorithm does have some limitations. When those limitations are exceeded, a skip reset will occur. This section describes the possible causes of the skip reset, and what can be done.

When a skip reset has occurred, along with changing the settings to prevent a future reset, it is recommended to perform a Reset Current Status operation in the Tools menu. This will clear the skipping data and set it to start at the beginning again.

The first cause of a skip reset is that the skip size is set to low or too high. The skipping is based on the read size per head, and that can vary between drives. The default skip size is 4096 LBA

which handles many cases. The following table gives the range of read size per head that the algorithm can handle with a given skip size.

skip size: 128 good from 1MB to 300MB

skip size: 256 good from 2MB to 500MB

skip size: 512 good from 5MB to 900MB

skip size: 1024 good from 9MB to 1700MB

skip size: 2048 good from 17MB to 3300MB

skip size: 4096 good from 35MB to 6500MB

skip size: 8192 good from 70MB to 13000MB

skip size: 16384 good from 150MB to 26000MB

skip size: 32768 good from 300MB to 50000MB

Samsung drives will commonly have a very high read size per head compared to other drives, and may require setting the skip size higher, maybe 16384. On rare occasion you may find a drive that has a very small read size per head. I have seen one ddrescue log where the read size per head appeared to be about 1MB.

The second most likely cause for a skip reset is that the skipping is happening due to slow reads caused by some sort of slow responding issue. Western Digital drives are commonly known for having a slow responding issue, and also occasionally some Seagates. If a skip reset happens and you notice that the slow skip count is high, then this may likely be the cause. The best solution is to obviously fix the slow responding issue. But if this is not possible, you can increase the Skip Threshold time to a high value such as 60000.

A third cause of skip resets is that the drive is not reading any data at all, and is returning error/abort for all reads. In this case the skip reset will be triggered very quickly, almost instantly after you start the cloning process. You would also get this same result if you selected ATA passthrough for a USB connected drive, as USB drives require SCSI passthrough.

If you are unable to determine the cause of the skip resets and which to continue the recovery without being stopped for the skip resets, you can disable Phase 1 and Phase 2 in the clone settings. If you are getting skip resets, those phases are not going to be productive anyway. Another alternative to dealing with skip resets is to set the max skip size to less or equal to 16 times the min skip size. This can be an effective way of still using phase 1 and 2 when getting skip resets, but the skip reset will not stop the program in this case. This is meant mostly for solid state drives and flash drives, where the head skipping algorithm does not work well, and the program should not stop just for a skip reset.

Relay Operation

This section explains how to setup and use a relay for power cycling the device.

For information about what relay to purchase, and how to wire it up, please see the instructions on the website here: http://www.hddsuperclone.com/sitev1/hddsuperclone/usbrelay

Once you have the relay ready, connect it to the computer via USB, and also connect it to the device, then start HDDSuperClone. Choose the project, mode, source, destination, and any other desired settings as normal. Then under the **Settings** menu, select **Choose Primary Relay Board**. This will list all supported relays that are currently connected to the computer, and allow you to select the desired relay. This will only list connected relays that are directly supported in HDDSuperClone.

Once you have selected the relay, under the **Settings** menu, select **Primary Relay Settings**. By default when the relay is activated, it will energize all relays for the activation time, then de-energize the relays, and wait for the post activation delay time before resuming and attempting to reconnect to the device. You may need to adjust the times for your device, the post activation delay time needs to be long enough for the drive to become ready after powering back on. It is recommended to use the **Test relay operation** button with the drive connected to make sure it works as expected. Changing the activation conditions of the relays is for advanced users. Once you have the settings correct, click OK.

Now in **Setting** menu select **Advanced Settings**. Under **Action to perform on major drive error**, choose **Activate primary relay.** This will activate the relay board every time the drive is locked up and not responding. Click OK to save the settings.

Virtual Disk Device Driver Mode (for data extraction)

This section explains what the Virtual Disk Device Driver Mode (for data extraction) is and how to use it.

Important notes:

Once you choose one of the virtual disk modes, it enables a domain file, and will start with an empty domain. The domain is updated with any attempts to read in the virtual modes. **Any future attempts in Clone Mode will be limited to the domain.** This allows targeting only the areas that were attempted during the virtual mode operations. **To resume the ability to clone the whole drive, choose to Clear Domain from the File menu.**

There is a checkbox setting on the main screen for Slow Driver, which is set by default. **When the driver is in slow mode, it WILL use more CPU, it WILL cause small system lockups that can last several seconds, and that combination will make it slower. Plus the small lockups will likely result in IO errors being returned instead of good data for whatever read it was locked up on.** But the slow mode is meant to help prevent a total system lockup

which would require a reboot. **It is sometimes best to use the slow mode when first starting to work with the disk. It is MANDATORY to use the slow mode when performing a PARTPROBE command** (and if you choose to attempt to use testdisk, you might also need to keep the driver in slow mode). The driver does not normally need to be in slow mode when using a good third party program such as R-Studio to access the virtual disk. **It is best to try to turn the slow mode off if possible.** But if you end up with a total system lockup, you will know that you will need to keep the slow mode on during whatever circumstance caused the lockup. At this time it is my belief that software that is using multi-threading to access the disk is the main cause of the system lockups. The virtual disk needs to only have single point access to be stable.

Update on Linux kernel versions with the virtual disk driver: The virtual driver seems to work well with Linux kernel 3.13 (Ubuntu 14.04), which is where I have done most of the testing. Kernel 4.18 (Ubuntu 18.04.2) seems to be unusable with the virtual driver, and just locks up no matter what. So somewhere between 3.13 and 4.18 some changes were made that do not work well with the virtual driver. The good news is that starting with kernel 5.0 (Ubuntu 18.04.3), and also 5.3 (Ubuntu 18.04.4), the virtual driver seems to work very well, even better than back at 3.13. I was able to use TestDisk without it locking up, and also a partprobe command did not lock up the system, although the partprobe command itself locked up and did not return to a command prompt, but the window could be closed, and the command worked. So if you have lockups, check your kernel version with the command uname -r.

The virtual disk device driver presents the recovery to the operating system as a generic block device, which can be accessed like a normal disk. The virtual disk cannot be written to (writes will fail), only read from. Any data requested from the virtual disk will be read from either the source or destination disk. If the data has been previously read, it will come from the destination. If that data has not been previously read, it will be read from the source, written to the destination, and then passed on to the operating system. This means that the same data is never requested from the source more than once. This mode also creates a domain file, which is a domain of all data that was requested. This allows targeting specific areas in the cloning process.

When you choose one of the virtual disk modes and click Start, the program will create the block device and present it to the operating system. At this time, the operating system tries to get the some initial data from the disk about the partitions, but it is not yet in a condition to report actual data. So it returns zeroed data to the operating system until it is ready. This causes the OS to think the drive is empty, and therefore it does not list the disk with any partitions. In a way this is good, as it keeps the OS from trying to read the disk. If you need the operating system to report the partitions, you can use the partprobe command. Note that this command will likely lock up the system for a short time, and the computer may seem unresponsive. This is because of how the program is doing things backwards to present a real drive as a virtual drive, and there is a timeout in the driver that allows getting out of the lockup condition. Be patient and wait several seconds, if not a few minutes for it to complete. If it does not complete in a few minutes, then the system is most likely locked up and will require a reboot.

When you are in the driver mode and you click Stop, it removes the virtual disk from the system. It is best to close any programs that are accessing or have accessed the virtual disk before

clicking on Stop. Otherwise those programs could still try to access the virtual disk after it is deleted, and could lockup or crash.

Virtual Disk Modes:

There are 5 virtual modes to choose from, and they can be changed on the fly. There is no skipping when using the virtual modes as it is copying the data exclusively requested, so it does not use phases 1, 2, or 3. When using any mode greater than mode 1, not all data is recovered at that time, and you would need to perform additional steps to read the most data possible.

Virtual Mode 1 – This mode will use all available phases (starting with phase 4) when reading from the source. If there is a read error in the data chunk being requested, it will be processed according to what phases are enabled in the clone settings.

Virtual Mode 2 – This mode only uses phase 4, and does not perform any further processing of a chunk if it failed to read using phase 4.

Virtual Mode 3 – This mode is the same as mode 2, except it will increase the read size (up to the cluster size) when it detects sequential reads. This is actually similar to what the operating system driver typically does. Some recovery software can tend to read in small chunks which can cause slow reading. This option can help with that, but it will cause some over reading of unneeded data. It could also be potentially useful for drives suffering from a "slow" condition, where many reads are slow and reading larger chunks makes more sense. The advanced option Virtual Disk minimum cluster size can be set for this mode so that there is a minimum chunk read size to also help prevent small reads.

Virtual Mode 4 – This mode only reads from the destination, and does not read any data from the source. This mode also does not require a source to be chosen. This can be useful if you have either already cloned the disk, or performed enough operations using the virtual driver that the needed data is likely present on the destination.

Virtual Mode 5 – This mode does not read any data at all. It instead returns zeroed data (all zeros) with no error. This mode is meant for more quickly creating a recovery domain. This allows for creating a domain where the desired data resides, so that it can be attempted in the cloning process. It can be better to use the cloning process so that reads are sequential, as when attempting data extraction the reads will jump around causing much head movement.

When reading data from the virtual disk in modes 1 through 4, any data that was unable to be read will return a read error to the operation system by default. This is the way for 3rd party data recovery software to determine which files are damaged. This reaction can be changed in the advanced settings. It can be changed to either return zero filled data, or mark filled data. Depending on the software you are using to attempt data extraction, you may need to change this setting to achieve the desired effect.

Changing the virtual disk logical sector size:

Normally the logical sector size for the virtual disk would be the same as the logical sector size of the source. But there are cases where you convert a USB disk to SATA, and while as USB it reported the logical sector size as 4096, as SATA it reports the size as 512. This can make it difficult to perform data extraction. The solution is to use the advanced settings option **Use physical sector size as logical for virtual disk**. This will use the physical sector size as the

logical sector size when presenting the virtual disk to the operating system. The physical sector size is determined by the **Block size** setting in the Clone settings. The block size is the number of logical sectors per physical sector. Some drives may get the physical sector size from the identify results, but some may not so they would need the setting changed manually. For example, in the Clone settings the sector size would be 512 (this is the typical logical sector size), and to set the physical size to 4096, you would set the block size to 8 (8 * 512 = 4096).

Basic use concept:

Start the HDDSuperClone process in virtual mode 1. Then open whatever 3rd party recovery software you wish to use. You can optionally try to access the files through the OS itself it you wish (you must run the partprobe command first for it to list partitions, and there may be stability issues, possibly needing to use the slow driver). You should turn off the "slow driver" whenever possible, or there will be unwanted errors and lockups.

There are a few commercial recovery programs that have Linux versions. I have found that DMDE works best with the virtual driver as it is already designed to work with bad sectors, as long as you set the proper settings, which will be explained in the videos referenced below. R-Studio has also worked okay in my testing, but it does not perform as well with the virtual driver as DMDE, and can miss getting data. I tried UFS Explorer and the newer Recovery Explorer, but they both failed to find a partition on a bad drive right away, and wanted to scan the whole drive. So I can't recommend them to use with the virtual driver. There is also the open source and free TestDisk, but in all my testing TestDisk kept crashing and just did not work well. You do not want to use anything like PhotoRec that scans the whole disk for files, that is something you would use after cloning the entire device. My recommendation: Use DMDE. You can try the free version with the limitations, but I recommend you get the proper paid version (if you are using it to recover customer data, then you need the pro version to meet the licensing requirements). Just make sure to activate it in Linux and not Windows, as a license for Windows will not work on Linux!

Use the 3rd party software to connect to /dev/sdv (the default name for the virtual drive created by HDDSuperClone). Extract the desired data with the software, during which time the data will also be copied from the source to the destination. Note that if the software is unable to detect partitions and files, then you will need to perform a regular clone and then likely a raw recovery search on the clone. Data extraction with the virtual driver is not possible if the file table is missing or badly damaged.

Examples for using the virtual disk:

Please refer to the Video Examples section of the website for further help in using the Virtual Disk Driver.
http://www.hddsuperclone.com/sitev1/hddsuperclone/videos

Progress Log Structure

All of the lines that begin with the character "#" are comment lines. Some of these lines are informational, and some contain configuration data. You should not alter any of these lines.

Any line that does not start with "#" is a data line. The data in the data lines is in hexadecimal format.

The first data line contains the current position in LBA, and current status. The possible status types are as follows:
0x0 = Copy phase 1
0x2 = Copy phase 2
0x6 = Copy phase 3
0x8 = Copy phase 4
0x10 = Trimming
0x20 = Dividing 1
0x22 = Dividing 2
0x30 = Scraping
0x40 = Retrying
0x7f = Finished

The rest of the data lines contain the current progress status of the recovery. There are three basic columns, and two additional status information columns. Each line contains information for a block.

The first column is the starting LBA of the block, and the second column is the size in LBA of the block. The third column is the status of the block. The possible status types are as follows:
0x0 = Non-tried
0x10 = Non-trimmed
0x20 = Non-divided
0x30 = Non-scraped
0x40 = Bad
0x7f = Finished

In the event that the Mark Bad Head option is checked, these values will have 0x80 added to them to indicate the bad head.

The fourth column may contain some additional information about two separate items, and is three bytes wide (1 word and 1 byte). The low order byte contains skipping information used by the program during the first two copy phases to adjust the head skipping algorithm. Forward skipping (phase 1) values begin with 0x80 and go up to 0xbf. Reverse skipping (phase 2) values begin with 0xc0 and go up to 0xff. These values indicate the size of skip runs. The beginning of the skip run will start with the low value and increase.

The high order word of the forth column contains information about other errors or conditions. In Generic mode a value of 1 indicates a read error, and any other value other than 0 indicates an error accessing the device. With Passthrough an error value other than 0 indicates that some other error happened such as a device fault or a host adapter error. The following are values for the direct modes. All values are in hex. Note that the low nibble is referenced as "x" as that part of the value can vary and the explanation is not for the end user.

2x – The device did not come ready, either busy or drq

3x – Soft reset time reached and a reset was performed

4x – Soft reset time reached and drive was unresponsive after reset

5x – General timeout reached or other fault

6x – Timeout when getting the data

7x – Wrong drive returned (master/slave)

8x – Reset performed due to unwanted control changes

1xx and greater – Unwanted control data change info

The fifth column will contain the return status of a bad read, along with the read time. It is 4 bytes wide, a 3 byte value for read status, and a 1 byte value for the read time. For the read status (high 3 bytes), in SCSI passthrough mode this will be 3 bytes. In order from left to right (high to low) those bytes are the sense-key, asc, and ascq. For the other ATA modes this will be 2 bytes. In order from left to right (high to low) those bytes are the status register, and the error register. With passthrough if the value in this column is a large number such as 0xffffff, it is an indication that there was an error and no sense data was present. In ATA passthrough mode there may not always be valid ATA return data. In this case the sense data will be used instead.

The low order byte contains the read time. By default, the time is in seconds, and there is no rounding. A value of 0 would indicate between 0 and 0.999 seconds, a value of 1 would indicate between 1 and 1.999 seconds, and so on, up to 63 (0x3f) seconds. The two high order bits are reserved for a possible future option for a higher time resolution.

520 Byte SAS Drives

Please note that odd size sector SCSI/SAS drive support is limited. I cannot guarantee that it will work in your case. Usually the reading works, but someone did report a case where they could not read from some drives. Writing is even more touchy, and may not work.

If you do not follow the instructions below correctly, you could end up trying to read/write the wrong sector size. You will get some sort of error when that happens, please check your settings if it is not working.

The Pro version of HDDSuperClone is capable of cloning/imaging 520 (and 528) bytes per sector SAS drives. These drives normally come from proprietary RAID units. Normally, the data is only 512 bytes, and the extra 8 bytes is extra data for the RAID, which means it is typically not needed for data recovery. It is possible to make a full image of a 520 byte drive with no setting changes. The sector size should be detected automatically, and the image file will contain the full sector data.

To access any SAS drive in Linux, you must have a NON-RAID card installed. A RAID card will not work, it must be NON-RAID so that disks can be accessed as individual drives. An example of a NON-RAID card is the SuperMicro AOC-SASLP-MV8.

To clone a 520 byte drive to another 520 byte drive requires enabling the advanced setting "Enable SCSI write to destination". PLEASE SEE THE NOTE IN THE IMPORTANT NOTES SECTION ABOUT USING THIS OPTION! This must be done to write to any odd sector size drive, as the OS is not capable of handling it. Note that from my testing, writing to an odd sector size drive can be much slower than normal, but if it is really slow, there may be an issue with writing, and the written data should be verified to make sure it is valid. To help the write speed, you can increase the cluster size, but do not go above 504! In my testing, bad things happened with a cluster size greater than 504.

To clone a 520 byte drive to a 512 byte drive (or image file) requires changing the advanced setting "Output sector size adjustment". To clone from a 520 byte disk to a 512 byte disk, you would change this setting to negative 8. This will trim the last 8 bytes from the 520 byte sectors to achieve 512 bytes.

To clone a 512 byte drive (or image file) back to a 520 byte drive would require enabling the SCSI write, along with changing the output sector size adjustment to positive 8. This will pad the 512 byte sectors with 8 zeros to achieve 520 bytes.

To clone a 520 byte image file back to a 520 byte drive would require enabling the SCSI write, and also changing the sector size to 520 BEFORE choosing the source image file. If you choose the source image before changing the sector size, you will get an error at the end of the recovery that the source size changed, or maybe a write error. This is because it thinks there are more sectors than there really are.

When the SCSI write is enabled, source and destination drives will list as sgX devices instead of sdX. This may help with writing in some cases. If things are not working, try making sure you are enabling the SCSI write before choosing the destination. It may also help with reading, the same applies, enable the SCSI write before choosing the source disk if it won't read.

If you are unable to write to a 520 byte drive with HDDSuperClone, there is always the possibility to use sg_dd from sg3_utils. But you are on your own for figuring out how to do that. I can only provide a starter command:

```
sg_dd if=/dev/sda of=/dev/sdb bs=520 blk_sgio=1
```

USB Direct Mode

The USB Direct mode is meant for USB only drives that cannot connect directly as SATA. If you can connect the drive via SATA, then you should do that. It is also meant for other storage devices such as USB flash drives and NVMe drives. You would need the appropriate USB adapter for the device. It is also meant for devices that have difficulty with the standard SCSI passthrough mode.

Everything relies on the USB bridge, and how it reacts. For ATA drives, the bridge may or may not support soft and hard resets. There is a special bulk only reset for USB attached mass storage devices. This reset should perform whatever reset is needed to get the device ready for the next command. In the case of ATA drives, it should perform either a soft or hard reset, if supported. In the case of other types of devices, it should perform whatever type of reset is supported. But you are completely at the mercy of the USB bridge of the device/adapter as to what kind of reset it will perform.

There is one issue that I cannot overcome with the direct USB mode. The system calls to perform the operations only like to transfer 16KB per packet, which results in extra overhead when transferring data. This is most noticeable when you are maxing out the USB speed, such as most USB2 devices. With USB3, if the device is only capable of transferring at about half the USB3 capable speed, then you would not notice any difference, but if you get into a fast NVMe drive, you would start to notice it. The only workaround for the slower speed is to turn up the cluster size. That will help, but still not get absolute full speed due to the overhead. But then again, if you are using this direct USB mode, then you already have a problem drive that does not work well with the SCSI passthrough mode, so you are not going for top speed.

There are two main benefits of the direct USB mode. The first is that it can perform the resets using timeouts. This can help by sending resets if the device has not responded within a short time (same concept as soft/hard resets for SATA drives). The second benefit is for slow responding drives, you can specify a very large cluster size up to 65528. The idea is that in between every read, the device is performing some background operation, causing every read to take a long time. By increasing the cluster size to a large value, you get much more data per read, thus increasing the overall read speed. I have very limited resources to test with, but overall this seems to work. The only issue would be if the device did not like to transfer that much data at once, which would hopefully result in an error so you don't think it succeeded.

The source drive will be listed by how the USB part of the device (the USB bridge) identifies itself. So you will not see the normal name of the drive in the source listing. There will also be two sets of numbers preceding the ID. The first set is the buss:device number which is given by the computer, the second set is the vendor-id:product-id of the device. The best way to identify

the desired drive is to list the sources at first without the device connected, and then again with the device connected. This will allow you to see which one to choose as the source. It is not recommended to have two of the same make and model devices connected to the computer at the same time, as this makes it difficult to determine which should be the source. Once the device is chosen, it will get the SCSI inquiry data (and ATA data if applicable) to get the model information, and that will be displayed in the GUI interface as the source information.

NOTE: When working with devices such as SD card readers, it will only list one device, but there will likely be multiple logical units (one for each different card slot). If multiple units are detected, a pop up window will appear after choosing the source asking which logical unit number to use. The only way to find the correct one is trial and error. Also, my experience with card readers has shown a great amount of variation in whether or not they work with the direct USB mode, and even whether or not they work with certain cards.

IMPORTANT: If you do need to have another similar (same vendor-id:product-id) USB drive connected, such as you are trying to clone the source to another similar USB drive, then it is important to connect the destination first, and then connect the source. It is a good idea to list (but don't choose) possible source drives after connecting the destination, and make note of the first numbers (buss:device) of the destination. Then connect the source and list the drives again. If is has the same buss number, the source will list above the destination with a higher device number, which is what you want for similar drives. If it has a different buss number, then it is okay, but otherwise the source should have a higher device number than the destination. Failure to do so could result in the destination being chosen as the source after a disconnection or power cycle.

When you choose the source drive in direct USB mode, the kernel driver is detached from the device, meaning it will disappear from the OS as any sort of /dev/sdX device. The program takes full control of the device, however if the device is power cycled or otherwise physically detached and reattached (USB connector unplugged and plugged back in), it will reappear to the OS, until such time as the program takes it back. To completely keep the OS from accessing the device, you can disable the USB mass storage driver in the Drives settings. This will prevent the OS from accessing any USB attached mass storage device, until such time as the driver is restored. This can help prevent the OS from causing the device to lock up, in the event that it is that unstable. The downside to this is that you can't use any other USB storage devices while the driver is disabled.

Disabling the USB mass storage driver performs the following actions:

```
cp -n /lib/modules/$(uname -r)/kernel/drivers/usb/storage/usb-storage.ko /root/usb-storage.ko.original
```

```
cp -f /lib/modules/$(uname -r)/kernel/drivers/usb/storage/usb-storage.ko /root/usb-storage.ko.backup
```

```
mv -fv /lib/modules/$(uname -r)/kernel/drivers/usb/storage/usb-
storage.ko /root/usb-storage.ko
```

Restoring the USB mass storage driver performs the following action:

```
mv -fv /root/usb-storage.ko /lib/modules/$(uname
-r)/kernel/drivers/usb/storage/usb-storage.ko
```

If you have already disabled or restored the driver, then performing the action again will result in a failure, as the file has already been moved.

The timer settings work a bit different. The initial busy wait time is used for the initial inquiry/identify command when choosing the device, and all read capacity commands, including when checking to see if the device has become ready again after an error. The busy wait time is used for the inquiry/identify command during operation when checking to see if the device has become ready again after an error. The soft reset time is still the main timeout to start the reset sequence if the device has not responded, this is the one you would change to adjust your timeout. The hard reset time is the timeout while waiting for the device to become ready after each reset type in the reset sequence, before moving on to the next reset type. You should not set this too low, I would recommend not going below 500ms. The reset timeout is used as the timeout for the last reset type in the sequence, to wait for the drive to become ready before going to the reset action.

There are 4 reset types. The first is the standard USB bulk only reset. This is the main reset for USB mass storage devices to tell the device to become ready for the next command. This should perform whatever reset the USB bridge is capable of for the device. It is recommend to leave this enabled as the first reset. Disabling it only disables it as the first reset, it is still used after every other reset type as it is the mandatory reset needed for USB mass storage devices. The second and third resets are passthrough ATA soft and hard resets. They are only for ATA drives, and may or may not be supported by the bridge. It is recommend to leave these disabled, they are only here in the event of some rare case where they work better than the initial bulk reset. The last reset is the port reset, which will reset the USB communication with the device. When this happens, the OS may temporarily try to access the device again before it is reclaimed by the program.

I don't have any USB only devices that have any errors on them, so all of my testing has been with USB adapters (the same thing I tell you not to use). So how it reacts with any particular device or drive is up to you to find out.

There are currently some functions that will output some data to the console for this mode. Some of this is meant to be console messages, and some is for debugging purposes. Don't read too much into what you see in the console. I will work on cleaning it up in the future. I just don't have the time right now.

Rebuild Assist

Rebuild Assist only works in Direct AHCI mode.

The Rebuild Assist Feature Set is an optional feature of the newer ATA standard. Its purpose is designed to help with rebuilding a RAID disk array when one disk is failing. When using the Rebuild Assist feature on a drive that supports it, the drive will perform a quick self scan to see if there are any bad read elements (heads). If it finds a bad head, it will mark it as such, which this documentation is calling "disabled". When trying to read from the bad head, it will report the LBA of the next good head, allowing skipping out of the bad head very rapidly. One other benefit of the Rebuilt Assist feature is that when enabled, it is supposed to minimize background activities. To find out if the Rebuild Assist feature is available on the drive, check the identify results after choosing the drive.

When Rebuild Assist is enabled, every time you "Connect" to the drive, it will enable the Rebuild Assist on the drive, and provide terminal output indicating what heads are enabled or disabled. During the Rebuild Assist phase, if the drive reports a predicted bad area (bad head or bad spot), you will see terminal output indicating a rebuild assist skip. When there is a rebuild assist skip, that area is marked as a bad head in the progress log. All other phases do not perform any skipping or marking based on the rebuild assist data. Choosing not to read the bad head in the first run will allow getting the most good data from the good heads first. Then a second run can be made to try to read any data possible from the bad head.

The proper way to use the Rebuild Assist on a drive that is capable is as follows.
1. Enable Rebuild Assist in the Advanced settings.
2. Disable Read Bad Head in the Clone settings.
3. Make sure the Rebuild Assist phase is checked in the Clone settings (it is checked by default).
4. Perform the recovery until it is finished.
5. If there are no Non-Tried blocks left indicating no rebuild assist skips., then the recovery is done as normal. If there are Non-Tried block left, continue with the next steps.
6. Enable Read Bad Head in the Clone settings.
7. Make sure the Rebuild Assist phase is unchecked in the Clone settings (it should have unchecked itself when the Rebuild Assist phase completed previously).
8. Perform the recovery again until it is finished, or as finished as desired, as it will be reading in the bad head at this time.

Tips and Tricks

Here are a few tips and tricks to help you along.

HDDSuperClone needs to be run as root. By doing so any files or folders that it creates will be owned by root. This can make it difficult to edit or delete files as a user (you will get permission errors). The terminal command "chmod" will help you with this. It is recommended that you read up on this command and learn how to use it. I try to do my best in the software to change permissions of any output files so that you shouldn't need to, but it is good to know how to use this command anyway.

There may be times when it would be nice to log all screen output to a file. Someone just recently pointed me to the "script" function. If you type "script –help" on the command line it will show the possible options. By default it will save the log to the file "typescript".
The following example would probably be the best way to use the function. This will log all screen output for that run of hddsuperclone to the file named hddsuperclone.log, and stop logging when hddsuperclone exits. If the file exists it will be overwritten.

```
script -c "sudo hddsuperclone" hddsuperclone.log
```

HDDSuperClone can be run from a live CD. When running from a live CD it runs in ram, and all data is lost when you reboot! You are responsible for making sure that you copy any data you wish to keep to an external drive when running from a live CD.

In the advanced and timer settings, commands can be run that are meant to run external relays of your choice. These are shell commands, just as if you were typing a command in the terminal. Commands can be separated by a semicolon. For instance, if you had a command to remove power from a drvie, and another command to power on, and you wanted a 5 second delay between the commands, and also wanted a 20 second delay after powering back on to allow time for the drive to spin up and become ready, your command may look something like this:

```
relayoff ; sleep 5; relayon ; sleep 20
```

Direct IO Restrictions

****** Note that direct IO is limited to IDE and PIO mode in the FREE version. ******

This program has two modes, passthrough and direct IO (either IDE or AHCI). Passthrough expects the drive is recognized by the Linux OS. When using passthrough you are using the Linux driver, so the driver will be aware of the activity and there should be no conflicts. However, if you choose to use direct IO mode, you are bypassing all Linux OS drivers! This could lead to conflicts, and undesired results if the drive was visible to the OS. HDDSuperClone will not let you select a drive from the menu that is visible to the OS. You must follow the method described in the section on how to hide a drive from the –OS.

In IDE mode it is also possible to work with a drive on the same controller as another (possibly a CD/DVD drive you are booting from). This is an extremely bad idea! HDDSuperClone will not

let you choose a drive if it thinks there is another drive on the same controller. At this time CD/DVD drives are not listed by model and serial, but will show up as patapi device. So if you see another device that has the same "ata" number, then there is another device on the same controller.

Also with IDE, if the device is a slave (device select is 1), there may be unexpected results with return status in the event of an error, or other unknown issues. HDDSuperClone will not let you choose a drive unless it is a master on the controller.

How to Hide a Drive from the OS

When using direct IDE mode (–direct) or direct AHCI mode (–ahci), the drive must be hidden from the OS for proper operation. The method below requires the Linux kernel to be 3.13 or newer. Ubuntu 14.04 has kernel 3.13, so it or any newer version of Ubuntu should work. If you are using a Linux distribution other than an Ubuntu flavor, you are responsible for knowing the kernel version.

Note that when using direct IDE mode with either actual IDE drives or SATA drives with the bios set for IDE, if you boot the OS without the drive plugged in / powered up and then connect it after booting, then Linux will not see it. This is an alternative method that seems to work the same, but it is still recommended to hide the drive.

Please note that when working with SATA drives and you want to use the Direct AHCI mode (recommended), you must set your computer BIOS to AHCI. If you have the BIOS set to IDE, no drives will be listed in Direct AHCI mode. Also, it has been reported that there can be an additional setting in BIOS for enabling or disabling the ability for hot plugging. This HOT PLUG also needs to be enabled.

The first thing you need to know is that this is based on the physical port that the drive is connected to. So once you disable that port you can plug any drive into it and Linux will not see it (this might not be entirely true for AHCI, as it would appear that Linux can tell when a device is plugged in and sends at least one command to it, likely an identify device command, but after that it leaves it alone). This is true for both IDE and AHCI. However, when working with SATA drives, switching between IDE and AHCI modes in the BIOS will change the port numbers and you would need to adjust accordingly. It is recommended to put the BIOS in AHCI mode when working with SATA drives. I have also once experienced the port numbers being different on one boot up, but it looked like something did not load properly on that boot, and a reboot put things back to normal. It is always a good idea that after the computer boots up, connect a good drive to the port to make sure it is still hidden.

The second thing you need is the number related to that port. To get this you need to plug in a good drive to that port and run HDDSuperClone. To get the port in the GUI, you must list the source drives. To do this you must create a new project (you can deleted it afterwards), then

switch the mode to Direct AHCI, then select to Choose Source Drive. You don't need to choose a drive, you just need to get it to list the drives with ports. Below is a sample output:

```
1) ata5.00 sda ST3120213AS 5LS3NJ70
2) ata6.00 sdb Hitachi HDS723020BLA642 MN3220F30JKX9E
3) ata7    --- no device
4) ata8    --- no device
5) ata9.00 sdg WDC WD2500BEAS-00URT0 WD-WXHY07017481
6) ata10   --- no device
```

Selection number 5 is the drive/port we wish to hide. You can see that the OS sees it as /dev/sdg. What we want is the number after the ata, which is 9.00. It is best to use only the main part of the port number, which in this case is 9. You must use that number to add a kernel boot option to grub, which in this case would be:
`libata.force=9:disable`

IMPORTANT NOTE: Don't disable the primary hard drive you are booting from! If you do that to your Linux installation it will not boot! You would then need to either boot into recovery mode, or from a live CD if you can't boot into recovery mode, and restore the original GRUB configuration.

To add a boot parameter in an Ubuntu live CD:
To get to the grub menu on an Ubuntu live CD you need to press any key as soon as the first small logo appears at the bottom of the screen during boot up. If the language selection comes up select your language and hit enter. Then hit F6 for Other Options, and when the options box pops up hit ESC. Now you should see a line on the bottom of the screen, just above the F-key options. Use the right arrow key to get a cursor flashing and make sure you are at the end of that line. Also make sure that Try Ubuntu without installing is highlighted above (up and down arrow keys to change). Add the boot parameter (such as libata.force=9:disable) to the end of the line, and hit enter. Wait for it to boot up.

It is possible to disable multiple ports. It is recommended that they be in order from lowest to highest. For example, to disable ports 9.00 and 10.00 the parameter would look like this:
libata.force=9:disable,10:disable

To temporarily add a boot parameter to a kernel of a Linux installation:
1) Start your system and wait for the GRUB menu to show (if you don't see a GRUB menu, press and hold the left Shift key right after starting the system).
2) Now highlight the kernel you want to use, and press the e key. You should be able to see and edit the commands associated with the highlighted kernel.
3) Go down to the line starting with linux and add your parameter libata.force=9:disable to its end.
4) Now press Ctrl + x to boot.

To make this change permanent using HDDSuperClone for a Linux installation:

1) In HDDSuperClone choose Disable Ports from the Drives menu.

2) Enter the number(s) of the port(s) that you wish to disable/hide as integers in the Set disabled ports field. If disabling multiple ports, it is best to have them in order from lowest to highest. For example, to disable ports 9.00 and 10.00, you would simply enter "9 10" without the quotes.

3) Click the Update ports button. You should see the new GRUB configuration information in the Updated ports section.

4) If the data looks correct, click OK, and then confirm the operation.\

5) Reboot the computer.

To make this change permanent for a Linux installation – manual method:

1) The first thing is to make a backup of the configuration. From a terminal run the following commands:

```
sudo cp -n /boot/grub/grub.cfg
/boot/grub/grub_hddsc_original_bakup.cfg

sudo cp -n /etc/default/grub
/etc/default/grub_hddsc_original_bakup
```

2) From a terminal run: `sudo gedit /etc/default/grub` and enter your password.

3) Find the line starting with GRUB_CMDLINE_LINUX_DEFAULT and append libata.force=9:disable to its end. For example:
`GRUB_CMDLINE_LINUX_DEFAULT="quiet splash libata.force=9:disable"`

4) Save the file and close the editor.

5) Finally, start a terminal and run: sudo update-grub to update GRUB's configuration file (you probably need to enter your password). A reboot is required for it to take effect.

On the next reboot, the kernel should be started with the boot parameter. To permanently remove it, simply remove the parameter from GRUB_CMDLINE_LINUX_DEFAULT and run sudo update-grub again.

To restore the original startup configuration using HDDSuperClone:

Note that this only works if there is a backup of the GRUB configuration files. When using HDDSuperClone to disable the ports, it will have made the backup, otherwise if you did it manually you would have needed to complete the first step of making the backup files.

1) In HDDSuperClone choose Restore original startup port configuration from the Drives menu.

2) Confirm the operation.

3) Reboot the computer.

To restore the original startup configuration – manual emergency method:

Note that this only works if there is a backup of the GRUB configuration files. When using HDDSuperClone to disable the ports, it will have made the backup, otherwise if you did it manually you would have needed to complete the first step of making the backup files.

1) Run the following commands in the terminal:

```
sudo cp -f /boot/grub/grub_hddsc_original_bakup.cfg
/boot/grub/grub.cfg

sudo cp -f /etc/default/grub_hddsc_original_bakup
/etc/default/grub
```

2) Reboot the computer.

After disabling the drive/port you should get a result like this:

```
1) ata5.00 sda ST3120213AS 5LS3NJ70
2) ata6.00 sdb Hitachi HDS723020BLA642 MN3220F30JKX9E
3) ata7    --- no device
4) ata8    --- no device
5) ata9    --- WDC WD2500BEAS-00URT0 WD-WXHY07017481
6) ata10    --- no device
```

Notice that there is no longer a decimal place after ata9, and there are dashes in place of sdg, but the drive is seen by HDDSuperClone as it is listed with model and serial. This drive/port is now hidden from the OS, and you may use HDDSuperClone to access it in AHCI mode.

Alternate method:

There is an alternative method that is not supported because it does not totally prevent the kernel from accessing the device, which leaves the possibility of causing a system lockup during cloning, or causing the device to lock up when connected. There has not been any long term testing of this method to verify how well it works. The following command will remove a device until a reboot or the device is disconnected and reconnected, where "sdx" is the device:

```
sudo echo 1 > /sys/block/sdx/device/delete
```


Reporting Bugs

It is always possible that there are programming errors in this software. It is also possible that there are errors and omissions in this documentation. If you report them, they will get fixed. If you don't report them, they will just stay the way they are and will not get fixed.

Report bugs to (sdcomputingservice@gmail.com)

Please include the version number of hddsuperclone. Please include any error messages, either as images or exact text, and console output. Please include the progress log (project) file.

HDDSuperTool Usage

HDDSuperTool is now built into HDDSuperClone. To run HDDSuperTool, use the --tool option when starting HDDSuperClone.

```
hddsuperclone --tool
```

Please note that all command examples should be changed from hddsupertool to hddsuperclone --tool

Hddsupertool must be run as root. If no arguments are present on the command line, it will start the passthrough menu mode.

If no drive is chosen, then all possible available drives will be listed with model and serial numbers. In passthrough mode, the drives will have the form of /dev/sdx, followed by model and serial. If the drive is not ATA (did not respond to identify device command), then the returned sense code will be shown instead of model/serial.

In direct mode, it will show driver and port information. From left to right: Driver, PCI bus ID, Registers port, Control port, Bus port, Device select, Model, Serial. If there was not a proper response from the identify device command, then there will be other information in place of the model and serial. "Busy or drq" will be followed by the status and error register values, and usually means there are no drives connected to that controller. "No drq" will be followed by the status register, lba mid, lba high, and either "no device" or "patapi device". Patapi indicates there is a CD/DVD drive attached. At this time hddsupertool does not list CD/DVD drives by model and serial.

If you would like to see more information about the ports on your system, try the command "lspci -v". Note that for SATA drives, the BIOS must be in IDE mode for direct IO to work. AHCI mode is not supported for direct IO.

Note that in direct IO mode, DMA access is not available in the free version.

The format for running hddsupertool is:

hddsupertool [*options*] *optional_variables*

Where:

optional_variables

Optional variables to be passed to the script from the command line. Both number and string variables can be passed, with a total limit of 16 variables that can be passed. A number variable is passed using a single equal sign, and a string variable is passed using a double equal sign. There must be NO SPACES BETWEEN THE VARIABLE NAME, EQUALS SIGN, AND VALUE. In the command line you also DO NOT START THE VARIABLE NAME WITH "$" as you would inside the script. Number variables will be treated as decimal unless preceeded by 0x in which case they will be treated as hex. String variables that contain text with spaces must be enclosed in quotes. The following example will set a number variable "number" to a value of 255 and a string variable "string" to a line of text.

hddsupertool -t /dev/sda -f script.txt number=0xff string=="line of text"

Hddsupertool supports the following options:

'-h'
'--help'

Print an informative help message describing the options and exit.

'-v'
'--version'

Print the version number and exit.

'-V'
'--verbose'

Show additional information. It is multi-level, meaning -V is level 1, -VV is level 2 and so on.

'-c'
'--check'

Perform the normal pre-check on the script, but do not execute it.

'-C'
'--command'

Use a command prompt to enter commands instead of reading from a script. Note that some commands will not work on the command line, and will report a generic error with the command (you can't do conditional statements, loops, or subroutines). When ran with this option you will get a command prompt of ">". To exit the command line mode type "end". The following example performs a simple identify device command and displays the raw results.

hddsupertool /dev/sda --command
hddsupertool 1.5-1.4 20160101

```
> buffersize 512
> setreadpio
> ata28cmd 0 0 0 0 0 0xa0 0xec
> printbuffer 0 0 512
0: 7a 42 20 08 00 00 10 00 00 00 00 00 3f 00 00 00    zB .........?...
10: 00 00 00 00 30 30 30 30 30 30 30 30 30 30 30 30    ....000000000000
20: 30 30 30 30 30 30 31 30 00 00 40 00 00 00 30 30    00000010.....00
30: 30 30 30 30 31 30 4d 56 61 77 65 72 56 20 72 69    000010MVawerV ri
40: 75 74 6c 61 49 20 45 44 48 20 72 61 20 64 72 44    utlaI EDH ra drD
50: 76 69 20 65 20 20 20 20 20 20 20 20 20 20 40 80    vi e          .
60: 00 00 00 2f 00 00 00 02 00 00 07 00 20 08 10 00    .../........ ...
70: 3f 00 00 fe 1f 00 10 01 00 00 20 00 00 00 07 00    ?......... .....
80: 03 00 78 00 78 00 a0 00 78 00 00 00 00 00 00 00    ..x.x...x.......
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
a0: 1e 00 17 00 08 40 08 40 00 41 08 40 08 00 00 41    .......A...A
b0: 07 04 00 00 00 00 fe 40 00 00 00 00 00 00 00 00    ...............
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
d0: 00 00 00 00 00 40 00 00 00 50 90 c2 9e fe 59 18    ........P...Y.
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
1a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
1b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
1c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
1d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
1e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
1f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a5 a4    ...............
> end
```

Total program run time = 80 seconds

'-d'
'--direct'

Enable and use direct IO mode. For SATA drives, the BIOS must be in IDE mode for this to work. It will not work in AHCI mode. Note that for some unknown reason, on some models of computers Linux may still use the AHCI driver even with the BIOS set to IDE.

'-a'
'--ahci'

Enable and use direct IO AHCI mode. The BIOS must be in AHCI mode for this to work.

'-f *script*'
'--file *script*'

The script file that contains the commands to be sent to the drive. If this is not set, then it will look for a file named "hddmenu" first in the current directory, then in the sub directory "hddscripts/", and then in "/usr/local/bin/hddscripts/". All subscripts will be loaded from the directory where hddmenu was found. If no script file is found then the program will exit.

'-i *spaces*'
'--indent *spaces*'

Performs an indentation cleanup on the script. This will indent certain commands by the number of spaces specified.

hddsupertool -i 2 -f script.txt

'-t *disk*'
'--target *disk*'

The drive to which the commands are to be sent. This must be a whole drive and not a partition. If you do specify a partition, all commands will still reference the drive from the beginning and ignore the partition offset. If this is not set, then a list of possible devices will be presented to choose from.

hddsupertool -t /dev/sda -f script.txt

'--ata'

List drives via ata-passthrough (default).

'`--scsi`'

List drives via scsi-passthrough.

'`Q`'
'`--quiet`'

Surpress some of the output. This is mostly for my use to be able to extract the help from the individual scripts to include in this documentation.

Example usage:

hddsuperclone --tool -t /dev/sda -f script.txt

HDDSuperTool Default Variables

There are several variables that are automatically created and used by the program. You should not use these variables to store data as they will get overwritten by the program during operation.

'`error_level`'

Some commands will return data in this variable. The data is command specific.

'`io_sense_key`'

sense key - part of the key code qualifier. Only for passthrough.

'`io_asc`'

additional sense code - part of the key code qualifier. Only for passthrough.

'`io_ascq`'

additional sense code qualifier - part of the key code qualifier. Only for passthrough.

'`io_status`'

This is the SCSI status byte as defined by the SCSI standard. Only for passthrough.

'`io_masked_status`'

Refer to "The Linux SCSI Generic (sg) HOWTO". Only for passthrough.

'io_msg_status'

Refer to "The Linux SCSI Generic (sg) HOWTO". Only for passthrough.

'io_sb_len_wr'

Number of bytes in the sense buffer. Only for passthrough.

'io_host_status'

Refer to "The Linux SCSI Generic (sg) HOWTO". Only for passthrough.

'io_driver_status'

Refer to "The Linux SCSI Generic (sg) HOWTO". Only for passthrough.

'io_resid'

Refer to "The Linux SCSI Generic (sg) HOWTO". Only for passthrough.

'io_duration'

Number of milliseconds the SCSI command took to complete. Only for passthrough.

'io_info'

Refer to "The Linux SCSI Generic (sg) HOWTO". Only for passthrough.

'ata_return_error'

ATA error register data.

'ata_return_count'

ATA count register data.

'ata_return_lba'

ATA LBA register data combined into one value.

'ata_return_device'

ATA device register data.

'`ata_return_status`'

ATA status register data.

'`time`'

Value returned from GETTIME command.

'`date`'

Value returned from GETTIME command.

'`data_transferred`'

Returns the number of bytes transferred. This can be an indication if data was transferred or not, but is not a guarantee that data was actually transferred.

'`command_status`'

Return value from a command. Any value other than 0 indicates the command failed to complete at the driver level, and any sense key or register data may not be present or useful. 1 is for passthrough only. 0x20 and up are for direct IO only. There are major and minor values that are combined to form the full value. Values are listed in hex.

Major values:
1 = Passthrough error (ioctl driver reported an error)
2x = Busy or DRQ timeout while waiting to send the command
3x = Soft reset timeout, command did not complete in time
4x = Soft reset timeout, drive is not ready after reset
5x = General timeout, command did not complete in time
6x = No DRQ, drq was not set when it was expected
7x = Wrong device detected, something switched the device
8x or higher = Something bad went wrong when processing
Minor values:
x1 = Exceeded reset timeout
x2 = Exceeded general timeout

'`bus_master_status`'

Bus master status. Only for direct mode when performing DMA commands.

'`direct_mode`'

A way for scripts to tell if using direct(value=1) or passthrough(value=0) mode.

'ahci_mode'

A way for scripts to tell if using AHCI mode.

HDDSuperTool Script Examples

The scripting can be a bit overwhelming to even those that are used to some sort of programming. This section is an attempt to demonstrate the basics.

The following is a commented script to perform a simple identify device command and display the raw data.

```
# first we must set the buffer size
buffersize 512

# now we must specify that this is a PIO read command
setreadpio

# now we send the command
ata28cmd 0 0 0 0 0 0xa0 0xec

# show the data on the screen
printbuffer 0 512

# write the data to the file "indentify.bin"
writebuffer "indentify.bin" 0 0 512

# the data returned from this command is in words in litte endian format,
# and can be flipped for easier viewing of the information
# note that this is optional and not normally used for other commands
wordflipbuffer 0 512

# show the flipped data on the screen
printbuffer 0 512
```

MHDD has several scripts available for different things. Here is an example of translating a MHDD script to HDDSuperTool. First here is a MHDD script to dump 2 sectors of module 02 from a WD Marvel drive (popular as the drive passwords are usually stored there).

```
;script name: read md
;reads md 02 on WD marwell drives
;
```

```
reset
waitnbsy

regs = $45 $0b $00 $44 $57 $a0 $80
waitnbsy

regs = $d6 $01 $be $4f $c2 $a0 $b0
waitnbsy
checkdrq
sectorsfrom = cs.bin

regs = $d5 $01 $bf $4f $c2 $a0 $b0
waitnbsy
checkdrq
sectorsto = 21.bin

regs = $d5 $01 $bf $4f $c2 $a0 $b0
waitnbsy
checkdrq
sectorsto = 22.bin

; end
```

This also requires the 512 byte file cs.bin to have been created by the user with the following data. Note that only the first line has data, the rest of the file is 0 filled.

```
0x000  08 00 01 00 02 00 00 00 00 00 00 00 00 00 00 00
....................................................
0x1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Here is the HDDSuperTool version.

```
# the first command is non-data, so set the buffersize to 0
buffersize 0

# this is a PIO command so this must be set
setreadpio

# send the first command
ata28cmd 0x45 0x0b 0x00 0x44 0x57 0xa0 0x80

# optionally show the returned values of the error and status registers
echo "error=0x" $ata_return_error " status=0x" $ata_return_status

# the next command sends 512 bytes (the file) so set the buffer size
buffersize 0x200
```

```
# clear the buffer to 0s so there is no other data
clearbuffer

# put data in the buffer
# note that this is the file data from cs.bin
# as you can see it can be directly entered with HDDSuperTool
setbuffer 0
  08 00 01 00 02
endbuffer

# this is a write command so set accordingly
setwritepio

# send the command
ata28cmd 0xd6 0x01 0xbe 0x4f 0xc2 0xa0 0xb0

# optionally show the returned values of the error and status registers
echo "error=0x" $ata_return_error " status=0x" $ata_return_status

# the next commands are read commands
setreadpio

# clear the buffer to 0s so we know the data received is good
clearbuffer

# send the command
ata28cmd 0xd5 0x01 0xbf 0x4f 0xc2 0xa0 0xb0

# optionally show the returned values of the error and status registers
echo "error=0x" $ata_return_error " status=0x" $ata_return_status

# write the sector to a file
writebuffer "21.bin" 0 0 0x200

# clear the buffer to 0s so we know the data received is good
clearbuffer

# send the command
ata28cmd 0xd5 0x01 0xbf 0x4f 0xc2 0xa0 0xb0

# optionally show the returned values of the error and status registers
echo "error=0x" $ata_return_error " status=0x" $ata_return_status

# write the sector to a file
writebuffer "22.bin" 0 0 0x200
```

this command is not listed in the MHDD script above
it is the counterpart to the first command
the first command turned on a special VSC mode
this command turns it back off, otherwise it would need to be reset
buffersize 0
setreadpio
ata28cmd 0x44 0x0b 0x00 0x44 0x57 0xa0 0x80

we are done
end

HDDSuperTool Script Commands

Note that all commands, operators, operands, and variables in the script MUST BE SEPARATED BY A SPACE. For example, "SETI $a=1" is not correct and will be rejected. The proper command would be "SETI $a = 1". This is the syntax that I used as it allows for less complicated programming code. Items can be separated by more than one space with no issues, it just has to be a minimum of one space.

Note that all numbers written in the script are assumed to be in decimal format, unless specified as hex by a proceeding "0x". The exeption is when setting the buffer all numbers are assumed in hex format.

All commands are available in all small letters or all capital letters, but they cannot be mixed small and capital letters. Commands are in all caps in this documentation for viewing purposes.

Most if not all commands will accept variables in place of actual numbers or text. If you find a command that does not properly accept variables when you think is should, please report it.

The available commands and syntax are as follows:

'#'

Comment line. Anything following the # will not be processed. It must be the first non-space character in the line. You cannot add a comment to the end of a line.

'ECHO'

Print to screen output. Text can be output using quotes, either single or double. Quotes of the opposite type inside the original quotes will be output to the screen. Both string and number variables can also be output. There must be a space between quotes and variables, and variables must be outside the quotes. The following is a proper example:

ECHO "This will print the variable a=" $a " and then the variable b=" $b
ECHO "This will print single 'quotes' within the line"

`'SETI'`

Set a number variable. All variables must start with "$". All number variables are integers of type long long. That means they can be a value from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. You can also set it to another variable, and also do simple math with two operands and one operator.

Available operands are:

+ addition
- subtraction
* multiplication
/ division (no remainder)
% division (result is the remainder)
& bitwise and
| bitwise or
^ bitwise exclusive or
< bitwise shift left
> bitwise shift right

Usage examples:

SETI $a = 3
SETI $b = $a + 0x10
SETI $c = $a * $b
SETI $count = $count + 1

There are a couple special cases for the SETI command. The first is the ability to extract a number from a string variable. The format is "SETI $number = $string location", where $string is the string variable containing the number to be extracted, and location is the placement of the number inside the string, starting with 0 as the first location. The location is based on spaces or tabs between the numbers. The location can be a number variable itself. If the number to be extracted starts with 0x it will be processed as hex, otherwise it will be processed as decimal unless forced using the "HEX" command. The following example will extract the number 3 from the string 1 2 3 4 5.

sets $string = "1 2 3 4 5"
SETI $number = $string 2

The second special case is extracting a number from the buffer or scratchpad (or sensebuffer, but words are not supported for sensebuffer). You can only extract a byte, word, double word, or quad word. The format is "SETI $number = BUFFER location type", where location is the starting byte location in the buffer, and type is either b, w, dw, qw. If no type is specified it will

assume the type is byte. Note that the words are little endian, so a buffer value of 11 22 33 44 would be double word value 0x44332211. This can be useful for extracting information from the identify device results.

```
#extract byte 234 from the buffer
SETI $number = BUFFER 234
```

```
#extract double word from the buffer starting at byte 234
SETI $number = BUFFER 234 dw
```

'SETS'

Set a string variable. All variables must start with "$". The format is the similar to the ECHO command, except instead of screen output the string ends up in the variable.

```
SETS $string = "This will put the variable a=" $a " and then the variable b="
$b " along with this message into a string variable"
```

There is one exception, where a part of the buffer can be extracted into the string variable. The format for that is "SETS $string = BUFFER location length", where location is the byte position in the buffer and length is the number of bytes to copy. The following example will put 40 characters from the buffer starting at byte 54 into a string variable:

```
SETS $string = BUFFER 54 40
```

'EXIT'

The script will terminate with an exit code. The format is "EXIT number", where number is a value from 0-255. Note that the program already uses some of the lower numbers for program error codes, so it is advised you use numbers 16-255.

'END'

The script will terminate with a normal exit code of 0.

'HEX'

This forces many commands to default to processing numbers as hex instead of decimal. It will affect the output of the ECHO command and also the SETS command. Be very careful using the HEX command, as it can cause undesired results if left turned on. It is recommended to only use it before a command such as ECHO to get the desired results from the command, and then use the DECIMAL command to turn it back off.

```
SETI $number = 0x7f
ECHO "The number in decimal is " $number
HEX
```

ECHO "The number in hex is 0x" $number
DECIMAL

`'DECIMAL'`

This turns off the HEX command and puts the default number processing back to decimal. It does not force numbers to be processed as decimal, if they are proceeded by 0x they are still processed as hex.

`'SETBUFFER'`

Manually set the buffer contents. The buffer contains the data that is either sent to or received from the drive. The format is "SETBUFFER offset", where offset is a number (can be a variable) that is the location in the buffer to start. The lines following the SETBUFFER command are the actual data, to be ended by the ENDBUFFER command. The data must be in the format of byte-space-byte-space-byte, and the data is always processed as hex. There is no set number of bytes per row, the next row will just pick up where the last row left off. A row can start with an additional offset (which will be an additional offset to the supplied buffer offset). To use the additional offset start the line with a hex number followed by a colon.

start setting the buffer at offset 128(0x80)
SETBUFFER 0x80
 0 1 2 3 4 5 6 7
 8 9 a b c d e f
 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
 # the next line will jump to buffer offset 256(0x100)
 80: 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
ENDBUFFER

`'ENDBUFFER'`

Required for ending the SETBUFFER command.

`'PRINTBUFFER'`

Prints the buffer to the screen. The format is "PRINTBUFFER offset size", where offset is the starting buffer offset, and size is the number of bytes to be printed. You cannot print past the end of the current buffer size.

show the first 256 bytes of the buffer
PRINTBUFFER 0 256

`'STRINGTOBUFFER'`

Puts a string into the buffer contents. The format is "STRINGTOBUFFER offset size $string", where offset is the buffer offset, size is the maximum copy size, and $string is the string variable

name. If size is greater than the length of the string then the whole string will be copied into the buffer, and the copy will stop at the end of the string. If size is smaller than the length of the string then it will only copy up to the size limit and the rest of the string is ignored.

```
SETS $string1 = "this is a string"
# copy the whole string (up to 64 bytes) to the buffer starting at offset 0x10
STRINGTOBUFFER 0x10 64 $string1
# copy only the first two words to the beginning of the buffer
STRINGTOBUFFER 0 7 $string1
```

'CLEARBUFFER'

Clears (erases) the entire buffer contents to zeros.

'BUFFERSIZE'

Sets a new buffer size. This erases (destroys) the current buffer contents. The format is "BUFFERSIZE size", where size is the new size of the buffer in bytes. The default starting buffer size is 512. The maximum buffer size is 33554432 (512 * 65536).

```
# increase the buffer size to 4096
BUFFERSIZE 4096
```

'WRITEBUFFER'

Write the buffer or part of the buffer to a file. The format is "WRITEBUFFER filename bufferoffset fileoffset size", where filename is the name of the file to write to, bufferoffset is the byte offset of the starting point in the buffer, fileoffset is the byte offset of the write point in the file, and size is the size in bytes to be written. If the file does not exist it will be created. This will overwrite the designated data in the file, but will not erase any other data in the file, meaning you can keep adding data to the file at different offsets.

```
# write the first 512 bytes of the buffer to the file named image.bin
WRITEBUFFER image.bin 0 0 512
# write 512 bytes to the second sector of the file
WRITEBUFFER image.bin 0 512 512
```

'READBUFFER'

Read a part of a file to the buffer. The format is "READBUFFER filename bufferoffset fileoffset size", where filename is the name of the file to read from, bufferoffset is the byte offset of the starting point in the buffer, fileoffset is the byte offset of the read point in the file, and size is the size in bytes to be read.

```
# read the first 512 bytes of the file named image.bin to the start of the
buffer
```

READBUFFER image.bin 0 0 512
read the second 512 bytes of the file to the buffer
READBUFFER image.bin 0 512 512

'DIRECTION'

Set the direction of the data transfer to either NONE, TO, FROM, or TOFROM. This is often required for proper processing of both SCSI and ATA passthrough commands by the Linux kernel. The proper direction needed for a command can be found in the SCSI or ATA documentation respectively. Note that for ATA commands this is set by the SETREADPIO, SETREADDMA, SETWRITEPIO, and SETWRITEDMA commands respectively, so you do not need to set it directly. You do need to set it with SCSI commands.

set the direction to read data from the drive
DIRECTION FROM

'SCSI6CMD'

Perform an scsi 6 command. Format is "SCSI6CMD b0 b1 b2 b3 b4 b5", where b0-b5 are the actual bytes of the command.

'SCSI10CMD'

Perform an scsi 10 command. Format is "SCSI10CMD b0 b1 b2 b3 b4 b5 b6 b7 b8 b9", where b0-b9 are the actual bytes of the command.

perform scsi capacity 10 command
DIRECTION FROM
SCSI10CMD 0x25 0 0 0 0 0 0 0 0 0

'SCSI12CMD'

Perform an scsi 12 command. Format is "SCSI12CMD b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11", where b0-b11 are the actual bytes of the command.

'SCSI16CMD'

Perform an scsi 16 command. Format is "SCSI16CMD b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15", where b0-b15 are the actual bytes of the command.

perform scsi capacity 16 command
DIRECTION FROM
SCSI16CMD 0x9e 0x10 0 0 0 0 0 0 0 0 0 0 0 0x12 0 0

'PRINTSENSEBUFFER'

Prints the SCSI return sense buffer to the screen so that advanced users can decode the results. The format is "PRINTSENSEBUFFER offset size", where offset is the sense buffer offset, and size is the number of bytes to be printed. The maximum sense buffer size is 256.

display the contents of the sense buffer that was returned, if any
PRINTSENSEBUFFER 0 $io_sb_len_wr

'SETREADPIO'

Prepare for PIO read. Required before performing this type of ATA command. If the buffer size is set to zero prior to this command, then it will setup for a non-data command. If you wish to perform a non-data command you should set the buffer size to zero before performing this command.

'SETWRITEPIO'

Prepare for PIO write. Required before performing this type of ATA command. If the buffer size is set to zero prior to this command, then it will setup for a non-data command. If you wish to perform a non-data command you should set the buffer size to zero before performing this command. The non-data command is the same as when performing a non-data SETREADPIO command.

'SETREADDMA'

Prepare for DMA read. Required before performing this type of ATA command.

'SETWRITEDMA'

Prepare for DMA write. Required before performing this type of ATA command.

'ATA28CMD'

ATA 28 bit command. The format is "ATA28CMD b0 b1 b2 b3 b4 b5 b6", where b0-b6 are the actual bytes of the command according to ATA documentation. Note that the device bit of the device field will be set for you according to the device being accessed, in either passthrough or direct IO modes. b0 = features b1 = count b2 = lba low b3 = lba mid b4 = lba high b5 = device b6 = command

perform ata_identify_device command
buffersize 512
setreadpio
ATA28CMD 0 0 0 0 0 0xa0 0xec
echo "Raw buffer:"
printbuffer 0 512

'ATA48CMD'

ATA 48 bit command. The format is "ATA48CMD w0 w1 w2 w3 w4 b5 b6", where w0-w4 are the actual words of the command according to ATA documentation, and b5-b6 are the device and commands bytes. The words are in standard format and not flipped (not little endian), meaning that 0x0001 has a value of 1. Note that the device bit of the device field will be set for you according to the device being accessed, in either passthrough or direct IO modes, so it does not matter what value you give that bit. Also remember that the LBA order is reversed from the 28 bit command. w0 = features w1 = count w2 = lba high w3 = lba mid w4 = lba low b5 = device b6 = command

```
# perform ata_identify_device command
buffersize 512
setreadpio
ATA48CMD 0 0 0 0 0 0xa0 0xec
echo "Raw buffer:"
printbuffer 0 512
```

'WRITELOG'

Write a string variable to a text file as a line. This will append the string to the end of the existing file. The format is "WRITELOG filename string", where filename is the name of the file to be written to, and string is the variable to be written. If the file does not exist it will be created.

```
# write text lines to the file named logfile.txt
sets $line1 = "This is the first line"
sets $line2 = "This is the second line"
WRITELOG logfile.txt $line1
WRITELOG logfile.txt $line2
```

'GETTIME'

This gets the current time and date and sets two variables. Number variable $time will have the time in microseconds since the Epoch (1970-01-01 00:00:00 +0000 (UTC)). String variable $date will contain the current date and time.

```
# display the time since Epoch and the current date
GETTIME
echo "time since Epoch usec: " $time
echo "todays date: " $date
```

```
# display the time elapsed
GETTIME
seti $start_time = $time
# do some program stuff here that takes time
GETTIME
seti $elapsed_time = $time - start_time
echo "time elapsed usec: " $elapsed_time
```

`'REOPENDISK'`

Only for passthrough, has no effect with direct IO. Closes and reopens the current disk. This seems to perform a soft reset on the drive, which can be helpful in certain cases where it may not respond properly after performing certain commands.

`'IF'`

Conditional if statement. The format is "IF value1 condition value2", where value1 and value2 are number or string variables and condition is the comparison to be made. The values can be actual numbers (or strings) or variables. If using an actual string (word), do not use quotes, and there cannot be spaces in the string. To compare strings with spaces in them you must set and compare string variables. Complex statements are not allowed (more than one condition). Math is also not allowed within the statement. Multiple IF statements can be nested. Each IF statement must be ended with the ENDIF command. Everything in between the IF and ENDIF is executed if the condition is true.

Available conditions are:

< less than
> greater than
= equal
!= not equal
<= less than or equal
>= greater than or equal
Note that only equal and not equal are allowed when comparing strings

Example:

```
seti $high = 100
if $a > 1
  echo "a is greater than 1"
  if $a > 10
    echo "a is greater than 10"
    if $a > $high
      echo "a is greater than " $high
    endif
  endif
endif
```

`'ENDIF'`

Marks the end of an IF statement.

`'ELSE'`

Statement to be processed when the IF and any ELSEIF statements are false.

Example:

if $a > 0
  echo "a is greater than 0"
else
  echo "a is less than or equal to 0"
endif

'ELSEIF'

Conditional statement to be processed when the IF or preceeding ELSEIF statements are false. The format is the same as the IF statement.

Example:

if $a > 10
  echo "a is greater than 10"
elseif $a < 0
  echo "a is less than 0"
elseif $a = 5
  echo "a is equal to 5"
else
  echo "a is between 0 and 10 but is not 5"
endif

'SUBROUTINE'

Marks the start of a subroutine. The format is "SUBROUTINE name", where name is the name of the subroutine. The subroutine must be ended with the ENDSUBROUTINE command.

subroutine show_ata_return
  echo "ATA return data:
  hex
  echo "error=0x" $ata_return_error " count=0x" $ata_return_count " lba=0x"
$ata_return_lba " device=0x" $ata_return_device " status=0x" $ata_return_status
  decimal
endsubroutine

'ENDSUBROUTINE'

Marks the end of a subroutine.

'GOSUB'

Calls a subroutine. The format is "GOSUB name", where name is the name of the subroutine. When the subroutine ends by either the ENDSUBROUTINE or RETURNSUB command, program execution will resume on the line followin the GOSUB command.

```
ata28cmd 0 0 0 0 0 0xa0 0xec
gosub show_ata_return
printbuffer 0 512
```

'RETURNSUB'

Returns from the current subroutine. This is meant to be used with conditional statements, to be able to exit the subroutine before reaching the ENDSUBROUTINE command.

```
subroutine do_something
  if $error_level != 0
    returnsub
  endif
  printbuffer 0 512
endsubroutine
```

'WHILE'

Conditional loop. The format is the same as the IF command. The WHILE command must be ended with the DONE command. If the WHILE condition is true then the code between the WHILE and DONE commands is executed. When the DONE command is reached, the WHILE condition is checked again to see if it is true or false. When the condition is false the code continues on after the DONE command, otherwise it loops back to the beginning of the WHILE command.

Example:

```
# count from 1 to 10
seti $count = 1
while $count <= 10
  echo "The count is " $count
  seti $count = $count + 1
done
```

'DONE'

Marks the end of the WHILE command.

'BREAK'

Breaks out of the current WHILE command. This is meant to be used with conditional statements, to be able to exit the loop without having to get all the way to the DONE command, or to exit when the condition is still true. Example:

```
# count from 1 to 10
seti $count = 1
# this while statement is always true so would loop forever without break
while 1 = 1
  echo "The count is " $count
  seti $count = $count + 1
  if $count > 10
    break
  endif
done
```

'SETSCRATCHPAD'

Manually set the scratchpad contents. The format and use is the same as the SETBUFFER command. The scratchpad is a second buffer that you can use. The difference between the buffer and scratchpad is that the scratchpad is not used for direct disk command data transfers. It is just for storing and manipulating any data you wish.

There is one important difference from the SETBUFFER command. The SETSCRATCHPAD command will allow for values greater than 255. When this happens, it sets a quad word in the scratchpad, instead of just one byte. This is in place to make it easier to write variable values to a file. So be careful with this!

'ENDSCRATCHPAD'

Required for ending the SETSCRATCHPAD command.

'PRINTSCRATCHPAD'

Print the scratchpad to the screen. The format and use is the same as the PRINTBUFFER command.

'STRINGTOSCRATCHPAD'

Puts a string into the scratchpad contents. The format and use is the same as the STRINGTOBUFFER command.

'CLEARSCRATCHPAD'

Clears (erases) the entire scratchpad contents to zeros.

'SCRATCHPADSIZE'

Sets a new scratchpad size. The format and use is the same as the BUFFERSIZE command.

`'WRITESCRATCHPAD'`

Write the scratchpad to a file. The format is and use is the same as the WRITEBUFFER command.

`'READSCRATCHPAD'`

Read a part of a file to the scratchpad. The format and use is the same as the READBUFFER command.

`'COPYSCATCHPADTOBUFFER'`

Copies a part of the scatchpad to the buffer. The format is "COPYSCATCHPADTOBUFFER scratchoffset bufferoffset size", where scratchoffset is the starting offset of the scratchpad, bufferoffset is the starting offset of the buffer, and size is the number of bytes to copy.

copy 512 bytes from scratchpad to buffer
copyscratchpadtobuffer 0 0 512

`'COPYBUFFERTOSCRATCHPAD'`

Copies a part of the buffer to the scratchpad. The format is "COPYBUFFERTOSCRATCHPAD bufferoffset scratchoffset size", where bufferoffset is the starting offset of the buffer, scratchoffset is the starting offset of the scratchpad, and size is the number of bytes to copy.

copy 512 bytes from buffer to scratchpad
copybuffertoscratchpad 0 0 512

`'VARIABLECHECK'`

Check if a variable has been set and what type it is. The format is "VARIABLECHECK variable", where variable is the name of the variable to be checked. The purpose of this is to check if a varaible was set properly on the command line. The result is returned in $error_level. A value of 0 = variable not set. A value of 1 = variable is of number type but only set in script. A value of 2 = varaible is of string type but only set in script. A value of 17 = variable is of number type and is set on the command line. A value of 18 = variable is of string type and is set on the command line.

check variable type for $input
variablecheck $input
if $error_level = 0
  echo "variable not set"
elseif $error_level = 1
  echo "variable is number but not set on command line"

elseif $error_level = 2
  echo "variable is string but not set on command line"
elseif $error_level = 17
  echo "variable is number and was set on command line"
elseif $error_level = 18
  echo "variable is string and was set on command line"
endif

`'GETFILESIZE'`

Get the size of a file. The format is "GETFILESIZE filename", where filename is the name (with optional path) of the file. Filename can be the raw file name or a string that contains the file name. The raw file name cannnot contain spaces. If there are spaces, the file name must be in a string. The size of the file in bytes is returned in $error_level. If the file does not exist or there is some other error then $error_level will contain -1.

```
# get the file size of the file "test.txt"
getfilesize test.txt
# get the file size of the file "the test.txt"
sets $filename = "the test.txt"
getfilesize $filename
if error_level = -1
  echo "File not found"
endif
```

`'DELETEFILE'`

Delete a file. The format is "DELETEFILE filename", where filename is the name (with optional path) of the file. Filename can be the raw file name or a string that contains the file name. The raw file name cannnot contain spaces. If there are spaces, the file name must be in a string.

```
# delete the file "test.txt"
deletefile test.txt
# delete the file "the test.txt"
sets $filename = "the test.txt"
deletefile $filename
```

`'CALLCOMMAND'`

Perform a command line command. The format is "CALLCOMMAND command", where command is the command line command to be performed. The command can be a string varaible, or a single word command. For commands that contain multiple words, you must use a string variable.

```
# perform the command "ls -a"
sets $command = "ls -a"
```

callcommand $command

`'USERINPUT'`

Get user input from keyboard. The format is "USERINPUT variable", where variable is the string variable where the user input is placed. This will pause the program waiting for the user to type something and then press enter.

Example:

```
echo "What would you like to do?"
echo "1) Perform action 1"
echo "2) Perform action 2"
echo "3) Perform action 3"
echo "Enter your choice:
userinput $choicestring
# extract the first number from the string
seti $choice = $choicestring 0
if $choice = 1
  echo "your choice was 1"
elseif $choice = 2
  echo "your choice was 2"
elseif $choice = 3
  echo "your choice was 3"
else
  echo "invalid choice"
endif
```

`'WORDFLIPBUFFER'`

Flip the bytes within words from little endian to normal. The format is "WORDFLIPBUFFER offset size", where offset is the offset in the buffer to start the flip and size is how many bytes to flip. If the size is an odd number the last byte will not be flipped. This command is useful for getting some proper readable information from the identify device command, such as the model and serial number.

```
# show the model and serial number of a drive
buffersize 512
protocol pio-data-in
direction from
ata28cmd 0 0 0 0 0 0xa0 0xec
wordflipbuffer 0 512
sets $model = buffer 54 40
echo "Model= " $model
sets $serial = buffer 20 20
echo "Serial= " $serial
```

'WORDFLIPSCRATCHPAD'

Flip the bytes within words from little endian to normal. The format and use is the same as the WORDFLIPBUFFER command.

'GETSTATUS'

This will update the status and error registers ($ata_return_error, $ata_return_status, and $ata_alternate_status). This only works for direct IO. For passthrough it attempts to send the proper protocol, but it would appear that Linux does not support that protocol and returns sense data of "invalid field in CDB".

'SOFTRESET'

This performs a soft reset of the device. Note that this will reset both devices on the controller. This only works for direct IO. For passthrough it attempts to send the proper protocol, but it would appear that Linux does not support that protocol and returns sense data of "invalid field in CDB". To perform a soft reset using passthrough, use the REOPENDISK command.

'USLEEP'

Sleep for x amount of microseconds. The format is "USLEEP microseconds". This is a way to make the program sleep without using CPU, perhaps waiting for a drive to become ready.

echo "Sleeping for 5 seconds"
USLEEP 5000000
echo "Done sleeping"

'SOFTTIMEOUT'

Only for direct IO, has no effect with passthrough. When performing an ATA command, the time to wait before sending a soft reset. The format is "SOFTTIMEOUT microseconds". The default is 15000000 (15s). This is used to automatically perform a soft reset if a command times out, so that it does not have to be done manually. If this happens, the register status will reflect the status after the soft reset. The soft reset waits for the drive to be ready up to the value of RESETTIMEOUT after performing the reset before returning. If you don't want to perform a soft reset when it times out, set the SOFTTIMEOUT value greater than the value of GENERALTIMEOUT.

'RESETTIMEOUT'

Only for direct IO, has no effect with passthrough. When performing an ATA command, the time to wait for the drive to be ready after a soft reset before returning. The format is "RESETTIMEOUT microseconds". The default is 15000000 (15s).

'BUSYTIMEOUT'

Only for direct IO, has no effect with passthrough. When performing an ATA command, the time to wait before giving up if the device is busy prior to issuing the command. The format is "BUSYTIMEOUT microseconds". The default is 15000000 (15s). If you don't want the device to time out for being busy, set this value greater than the value of GENERALTIMEOUT.

'GENERALTIMEOUT'

This timer is mostly for direct IO, but can also influence passthrough in some circumstances. When performing a command, the time to wait before giving up if the command has not finished. The format is "GENERALTIMEOUT microseconds". The default is 30000000 (30s).

'LOADSCRIPT'

Loads a different script into memory. The format is "LOADSCRIPT script", where script is the script file to be loaded. This is mostly designed for a menu driven system. All current variables will be passed on to the new script. The new script will start from the beginning.

'PREVIOUSSCRIPT'

Loads the previous script into memory, if there was one. This is designed for a menu driven system. When returning to the previous script, the previous script will start from the beginning. All current varaibles will be passed on to the previous script.

'INCLUDE'

This will append another script file to the end (bottom) the current one in memory. The format is "INCLUDE script", where script is the script file to be loaded. This is useful for including common subroutines without having to copy them into every script.

'UPLINE'

Used for display purposes. Moves the cursor up one line. This is useful for repeating data on the screen without scrolling.

HDDSuperTool Scripts

This section contains all the help available for the individual scripts.

ata28_erase_sectors_pio

Erase sector(s) from the disk using 28 bit write pio data-out command.

This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 256).
Example: hddsupertool -t /dev/sda -f ata28_write_sectors_pio sector=0 count=1

ata28_makebad

Make a bad sector using the old 28 bit writelong command.
This command is obsolete and not supported on all drives.
This requires number variable "sector" to be set.
"sector" is the starting sector to write.
Example: hddsupertool -t /dev/sda -f ata28_makebad sector=0

ata28_read_sectors_dma

Read sector(s) from the disk using 28 bit read dma data-in command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to read.
"count" is the number of sectors to read (max 256).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be written to.
Example: hddsupertool -t /dev/sda -f ata28_read_sectors_dma sector=0 count=1
file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata28_read_sectors_dma sector=2048 count=16
file=="sectors2048-2053.bin"

ata28_read_sectors_pio

Read sector(s) from the disk using 28 bit read pio data-in command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to read.
"count" is the number of sectors to read (max 256).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be written to.
Example: hddsupertool -t /dev/sda -f ata28_read_sectors_pio sector=0 count=1
file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata28_read_sectors_pio sector=2048 count=16
file=="sectors2048-2053.bin"

ata28_readlong

Perform the old 28 bit readlong command.
This command is obsolete and not supported on all drives.
This requires number variables "sector" to be set.
"sector" is the sector to read.
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be written to.
Example: hddsupertool -t /dev/sda -f ata28_readlong sector=0 file=="longsector0.bin"

ata28_write_sectors_dma

Write sector(s) to the disk using 28 bit write dma data-out command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 256).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be read from.
Example: hddsupertool -t /dev/sda -f ata28_write_sectors_dma sector=0 count=1 file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata28_write_sectors_dma sector=2048 count=16 file=="sectors2048-2053.bin"

ata28_write_sectors_pio

Write sector(s) to the disk using 28 bit write pio data-out command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 256).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be read from.
Example: hddsupertool -t /dev/sda -f ata28_write_sectors_pio sector=0 count=1 file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata28_write_sectors_pio sector=2048 count=16 file=="sectors2048-2053.bin"

ata48_erase_sectors_pio

Erase sector(s) from the disk using 48 bit write pio data-out command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 65536).
Example: hddsupertool -t /dev/sda -f ata48_write_sectors_pio sector=0 count=1

ata48_read_sectors_dma

Read sector(s) from the disk using 48 bit read dma data-in command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to read.
"count" is the number of sectors to read (max 65536).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be written to.
Example: hddsupertool -t /dev/sda -f ata48_read_sectors_dma sector=0 count=1 file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata48_read_sectors_dma sector=2048 count=16 file=="sectors2048-2053.bin"

ata48_read_sectors_pio

Read sector(s) from the disk using 48 bit read pio data-in command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to read.
"count" is the number of sectors to read (max 65536).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be written to.
Example: hddsupertool -t /dev/sda -f ata48_read_sectors_pio sector=0 count=1 file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata28_read_sectors_pio sector=2048 count=16 file=="sectors2048-2053.bin"

ata48_write_sectors_dma

Write sector(s) to the disk using 48 bit write dma data-out command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 65536).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be read from.
Example: hddsupertool -t /dev/sda -f ata48_write_sectors_dma sector=0 count=1 file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata48_write_sectors_dma sector=2048 count=16 file=="sectors2048-2053.bin"

ata48_write_sectors_pio

Write sector(s) to the disk using 48 bit write pio data-out command.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 65536).
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be read from.
Example: hddsupertool -t /dev/sda -f ata48_write_sectors_pio sector=0 count=1 file=="sector0.bin"
Example: hddsupertool -t /dev/sda -f ata48_write_sectors_pio sector=2048 count=16 file=="sectors2048-2053.bin"

ata48_write_uncorrectable

Make bad sector(s) from the disk using 48 bit write uncorrectable non-data command.
This creates a pseudo-uncorrectable error without logging.
This requires number variables "sector" and "count" to be set.
"sector" is the starting sector to write.
"count" is the number of sectors to write (max 65536).
Example: hddsupertool -t /dev/sda -f ata48_write_uncorrectable sector=0 count=1

ata_disable_look_ahead

Disable read look ahead.
Example: hddsupertool -t /dev/sda -f ata_disable_look_ahead

ata_enable_look_ahead

Enable read look ahead.
Example: hddsupertool -t /dev/sda -f ata_enable_look_ahead

ata_identify_device

This will send the identify device command to the drive and display the raw results.
It will also display some of the drive capabilities and status.
A value of 0 = not supported or not enabled.
A value of 1 = supported or enabled.
Example: hddsupertool -t /dev/sda -f ata_identify_device

ata_scan_device

This will scan the drive using the read verify command.
Reads are done in blocks (clusters) of 256 sectors.
This means that all warnings and errors show the position
 of the first sector of the block, NOT the actual problem sector.
If the required variables below are not set on the command line,
 then you will be propted to enter them.
This requires the number variable startsector to be set.
This requires the number variable endsector to be set.
If endsector is set to 0, it will default to the last addressable sector.
This uses the identify device command to determine the last sector.
This requires the number variable threshold to be set.
Threshold is time in milliseconds.
If a read exceeds the threshold, a warning will be given.
If threshold is set to 0, it will default to 150ms.
This requires the number variable softtimeout to be set.
Softtimeout is the time in milliseconds to wait before giving up on a command.
If softtimeout is set to 0, it will default to 10000ms.
This requires the number variable hardtimeout to be set.
Hardtimeout is the time to wait after a softreset before doing a hardreset.
If hardtimeout is set to 0, it will default to 10000ms.
This requires the string variable logfile to be set.
Logfile will capture all the warnings and errors.
This requires the string variable rateslog to be set.
Rateslog will capture timing info that can be ploted.
Rateslog captures the low, high, and average over 256 block reads.
If logfile and/or ratesfile are blank no log will be produced.
Example using defaults:
hddsupertool -t /dev/sda -f ata_scan_device threshold=0 softtimeout=0 startsector=0 endsector=0
logfile=="" rateslog==""

ata_sct_error_control_timer

This command is not supported by all drives.
View the results of the identify device command to check support.
Get the value of the read error control timer.
Optionally set the value of the read error control timer.
This command uses the number variable "timer".
Timer can be a value from 0-65535.
The timer controls how much time before giving up on a read error.
Normally this is in 100ms increments (value of 5 would be 500ms),
 but this is drive specific (try different values to see what happens).
The default value is 0 (unlimited, all recovery proceedures are tried).
This value is not permenant, it resets with a power cycle.
If timer is not set it will only show the current value.
Example to show current error timer:
  hddsupertool -t /dev/sda -f ata_sct_error_control_timer
Example to set the error control timer to 1:
  hddsupertool -t /dev/sda -f ata_sct_error_control_timer timer=1

ata_sct_readlong

Perform an SCT read long command if supported.
This command requires the number variable "sector" to be set on the command line.
This also requires the string variable "file" to be set.
"file" is the name of the file the data will be written to.
Note that even if this command is supported, it may still be of no value.
The ATA documentation states that the data returned may be encoded.
And my experience so far is that the data is usually encoded and totally useless.
So don't get your hopes up about using this command to get any worthwhile data.
It is mainly a demonstration of using the SCT commands.
Example: hddsupertool -t /dev/sdb -f ata48_sct_readlong sector=100 file=="sector100.bin

ata_security_remove_password

Remove the password from the disk using 28 bit security disable password command.
This requires either the string variable "password" or "hex" to be set.
Password is the password in plain text.
Hex is the password in hex. It does not require 0x to proceed numbers.
If neither password or hex is set then the password will be blank (all 0s).
DO NOT SET BOTH or you could end up with undesired results!
This requires the number variable "type" to be set on the command line.
Type is either 0 for user password or 1 for master password.
If type is not set it will default to 0.
Example: hddsupertool -t /dev/sdb -f ata_security_remove_password password=="abcd" type=0
Example: hddsupertool -t /dev/sdb -f ata_security_remove_password hex=="0a 0b 0c 0d" type=0

ata_security_set_password

Set the password from the disk using 28 bit security disable password command.
This requires either the string variable "password" or "hex" to be set.
Password is the password in plain text.
Hex is the password in hex. It does not require 0x to proceed numbers.
If neither password or hex is set then the password will be blank (all 0s).
DO NOT SET BOTH or you could end up with undesired results!
This requires the number variable "type" to be set on the command line.
Type is either 0 for user password or 1 for master password.
If type is not set it will default to 0.
This requires the number variable "level" to be set on the command line.
Level is either 0 for high security or 1 for maximum security.
If level is not set it will default to 0.
Example: hddsupertool -t /dev/sdb -f ata_security_set_password password=="abcd" type=0 level=0
Example: hddsupertool -t /dev/sdb -f ata_security_set_password hex=="0a 0b 0c 0d" type=0 level=0

ata_security_unlock

Unlock the disk using 28 bit security unlock pio data-out command.
This requires either the string variable "password" or "hex" to be set.
Password is the password in plain text.
Hex is the password in hex. It does not require 0x to proceed numbers.
If neither password or hex is set then the password will be blank (all 0s).
DO NOT SET BOTH or you could end up with undesired results!
This requires the number variable "type" to be set on the command line.
Type is either 0 for user password or 1 for master password.
If type is not set it will default to 0.
Example: hddsupertool -t /dev/sdb -f ata_security_unlock password=="abcd" type=0
Example: hddsupertool -t /dev/sdb -f ata_security_unlock hex=="0a 0b 0c 0d" type=0

ata_smart_data

This will read the smart data from the drive and display the results.
Example: hddsupertool -t /dev/sda -f ata_smart_data

ata_status

This will get the current drive status registers if direct IO.
This does not work with passthrough.

good_subroutines

This contains common subroutines that are used by other scripts.
This file is required by most scripts.

hddmenu

Main menu
q) Quit
h) Toggle script help
1) Device information
2) Read sectors
3) Write sectors
4) Erase sectors
5) Tools
6) Security
7) VSC
8) Custom

hddsubmenu_custom

Custom menu
q) Quit
p) Previous menu
h) Toggle script help
1) Custom message
8) my scripts tech
9) my scripts test

hddsubmenu_erase

Erase menu
q) Quit
p) Previous menu
h) Toggle script help
1) Erase sectors 28 PIO
2) Erase sectors 48 PIO

hddsubmenu_info

Device information menu
q) Quit
p) Previous menu
h) Toggle script help
1) Identify device
2) Smart info

hddsubmenu_read

Read sectors menu

q) Quit
p) Previous menu
h) Toggle script help
1) Read sectors PIO 28
2) Read sectors PIO 48
3) Read sectors DMA 28
4) Read sectors DMA 48
5) Readlong (old 28 bit)
6) Readlong (48 bit SCT)

hddsubmenu_security

Security menu
q) Quit
p) Previous menu
h) Toggle script help
1) Security remove password
2) Security set password
3) Security unlock

hddsubmenu_tools

Tools menu
q) Quit
p) Previous menu
h) Toggle script help
1) Scan device
2) Make a bad sector
3) Write uncorrectable
4) Disable read look ahead
5) Enable read look ahead
6) Error Control Timer (SCT)
7) Reset Device
8) Get device status
9) Hard Reset

hddsubmenu_vsc

VSC menu
q) Quit
p) Previous menu
h) Toggle script help
1) WD dump mod 42 (older Caviar drives)
2) WD royl (Marvel) dump mod 02

3) WD royl (Marvel) dump mod 32
4) WD royl (Marvel) patch mod 02 (slow fix)
5) WD royl (Marvel) patch mod 32 (slow fix additional)
6) WD royl (Marvel) dump all modules
7) WD royl (Marvel) dump selected module
8) WD royl (Marvel) read rom
9) WD royl (Marvel) check rom file
10) WD royl (Marvel) write rom (dangerous)
11) WD royl (Marvel) write module (dangerous)

hddsubmenu_write

Write sectors menu
q) Quit
p) Previous menu
h) Toggle script help
1) Write sectors PIO 28
2) Write sectors PIO 48
3) Write sectors DMA 28
4) Write sectors DMA 48

hddsuperclone_debug.log

ERROR! Command 'HDDSuperClone' on line '1' not a valid command.
HDDSuperClone debug file created at Mon Dec 31 16:02:06 2018
There was an error processing the script 'hddsuperclone_debug.log'. Exiting...

reset

This will soft reset the device if direct IO.
If passthrough it will reopen the device.

reset_hard

This will hard reset the device if direct IO.
If passthrough it will reopen the device.

wd_dump_mod42

Western Digital (older) dump module 42 to file using vendor specific commands.
This will dump the first 3 sectors of the module to the file "mod42.bin".
It will also show it on the screen.

wd_royl_check_rom

Western Digital ROYL check ROM file.
This script performs some checks on a ROM file.

This requires the text variable "file" to be set.
"file" is the name of the file containing the ROM.
This scipt does not perform any disk IO,
so it can be ran with any target, even /dev/zero.

wd_royl_dump_mod02

Western Digital ROYL dump module 02 to file using vendor specific commands.
By default this will dump the sectors of the module to the file "mod02.bin".
It will also display the data on the screen.
Note that some USB drives do not support these vendor specific commands.

wd_royl_dump_mod32

Western Digital ROYL dump module 32 to file using vendor specific commands.
By default this will dump the sectors of the module to the file "mod32.bin".
It will also display the data on the screen.
Note that some USB drives do not support these vendor specific commands.

wd_royl_dump_mod_all

Western Digital ROYL dump all modules using vendor specific commands.
This will dump the sectors of the modules to the files "modulexxxx.bin".
Note that some USB drives do not support these vendor specific commands.

wd_royl_dump_mod_select

Western Digital ROYL dump a module to file using vendor specific commands.
By default this will dump the sectors of the module to the file "mod0x(select).bin".
It will also display the data on the screen.
This requires the number variable "mod" to be set in DECIMAL.
To enter a hex number preceed it with "0x".
mod is the SA module to be read.

wd_royl_patch_mod02

Western Digital ROYL patch module 0x02 using vendor specific commands.
This is to resolve a common WD slow issue, where the drive reads slow.
Definition of the "slow responding issue":
 - The drive is reading very slow, but all reads are GOOD.
This may not help any if the reads are bad due to a weak or bad head.
WARNING: THIS IS DANGEROUS AND COULD KILL THE DRIVE!!!
USE AT YOUR OWN RISK!!!
Western Digital ROYL patch module 02 to file using vendor specific commands.
Note that some USB drives do not support these vendor specific commands.
This patch is meant for drives that suffer from the "slow" issue.
This patch disables the background scanning on the drive.
This patch is based on data from the following forum thread:

WD - Fixing the "Slow Issue" manually :
http://www.hddoracle.com/viewtopic.php?f=86&t=848
This will dump the sectors to the file "<serial>_mod02orig.bin".
And it will create the file "<serial>_mod02patched.bin".
It will also create a folder using the model and serial as the folder name,
 and create a backup dump file with the date and time as part of the name.
It will create a backup dump every time a read or write is attempted.
The drive must be power cycled for the changes to take effect.
You have three options:
1) Read the module to a file and create the patch.
2) Write the patched data back to the disk.
3) Restore the original dump.

wd_royl_patch_mod32

Western Digital ROYL patch module 0x32 using vendor specific commands.
This is to resolve a common WD slow issue, where the drive reads slow.
Definition of the "slow responding issue":
 - The drive is reading very slow, but all reads are GOOD.
This may not help any if the reads are bad due to a weak or bad head.
WARNING: THIS IS DANGEROUS AND COULD KILL THE DRIVE!!!
USE AT YOUR OWN RISK!!!
This patch may not be needed if the module 0x02 patch fixes the slow issue.
You should perform the module 0x02 patch first and power cycle the drive.
If the module 0x02 patch works, then do not bother with this patch.
Western Digital ROYL patch module 32 to file using vendor specific commands.
Note that some USB drives do not support these vendor specific commands.
This patch is meant for drives that suffer from the "slow" issue.
This patch clears some data in the RE-LO list.
However, it may not clear everything that is needed, and is a bit of a hack.
It is intended be used along with the module 02 patch.
However, it is normally not needed if the module 02 patch solves the slow issue.
This patch is based on data from the following forum thread:
 WD - Fixing the "Slow Issue" manually :
http://www.hddoracle.com/viewtopic.php?f=86&t=848
This will dump the sectors to the file "<serial>_mod32orig.bin".
And it will create the file "<serial>_mod32patched.bin".
It will also create a folder using the model and serial as the folder name,
 and create a backup dump file with the date and time as part of the name.
It will create a backup dump every time a read or write is attempted.
The drive must be power cycled for the changes to take effect.
You have three options:
1) Read the module to a file and create the patch.
2) Write the patched data back to the disk.
3) Restore the original dump.

wd_royl_read_rom

Western Digital ROYL read ROM to file using vendor specific commands.
Note that some USB drives do not support these vendor specific commands.
This requires the text variable "file" to be set.
"file" is the name of the file to dump the ROM.

wd_royl_write_mod

Western Digital ROYL write module from file using vendor specific commands.
WARNING: THIS IS DANGEROUS AND COULD KILL THE DRIVE!!!
USE AT YOUR OWN RISK!!!
It is IMPORTANT to use 0x to select the module number in HEX!
The difference between decimal and hex COULD KILL THE DRIVE!!!
This modifies data in the service area of the drive!
If you don't know what you are doing, then don't do it!
Western Digital ROYL write module from file using vendor specific commands.
Note that some USB drives do not support these vendor specific commands.
WARNING! THIS SCRIPT COULD BE DANGEROUS IF YOU DO NOT KNOW WHAT
YOU ARE DOING!
It will create a folder using the model and serial as the folder name,
 and create a backup dump file with the date and time as part of the name.
It will create a backup dump every time a read or write is attempted.
The checksum will be recalculated if it not correct.
This requires the number variable "mod" to be set in DECIMAL.
To enter a HEX number preceed it with "0x" (IMPORTANT TO SELECT CORRECT
MODULE).
"mod" is the SA module to be written.
This requires the text variable "file" to be set.
"file" is the file which is to be written to the module.

wd_royl_write_rom

Western Digital ROYL write rom from file using vendor specific commands.
WARNING: THIS IS DANGEROUS AND COULD KILL THE DRIVE!!!
THIS IS EVEN MORE DANGEROUS OVER USB!!!
DO NOT STOP THIS PROCESS OR REMOVE POWER FROM THE DRIVE!!!
IF THE ROM GETS ERASED BUT NOT WRITTEN, DO NOT REMOVE POWER!!!
USE AT YOUR OWN RISK!!!
If you write an incompatible ROM you could kill the drive!
If the erase works but the write fails, you could kill the drive!
In this emergency case, use slow and stupid mode to attempt to write.
Do not use slow and stupid mode unless you erased the rom and are stuck.
It is recommended to perform another ROM dump and check after writing
 to be sure it worked. Use the other scripts for these functions.
If the script fails for any reason, it is recommended to do another dump
 and check to make sure ROM is still in good condition.
This requires the text variable "file" to be set.
"file" is the name of the file to dump the rom.

This requires the number variable "slow_and_stupid" to be set.
"slow_and_stupid" is an emergency mode if the flash is erased but not written.
A value of 0 is off, any other value is on.
It is not recommended to use slow_and_stupid unless needed.