

pdp11xfer

User's Manual

A command-line tool for transferring disk images between
a Linux host and a DEC PDP-11 over a serial line

Copyright ©2026 Large Scale Systems Museum
By Dave McGuire <mcguire@lssmuseum.org>

Version 1.0

Contents

1	Introduction	2
2	Requirements	2
3	Connecting to the PDP-11	2
3.1	Real hardware	2
3.2	SimH-simulated PDP-11s	3
4	Command-Line Reference	3
4.1	Synopsis	3
4.2	Modes	4
4.3	Connection options	4
4.4	Driver loading options	4
4.5	Device options	5
4.6	Other options	5
4.7	Exit status	5
5	Typical Workflow	6
5.1	The progress display	6
5.2	Examples	7
5.3	A complete SimH example	7
6	The PDP-11-Side Driver Programs	10
7	Troubleshooting	11
7.1	No driver detected	11
7.2	Controller/device mismatch	11
7.3	Timeouts and communication errors	11
7.4	Loader-specific errors	12

1 Introduction

pdp11xfer is a command-line program that transfers disk images between a Linux host and a DEC PDP-11 over a serial connection. It works with a variety of PDP-11 disk controllers and drives, and does not require any permanent software installation on the PDP-11: each time it runs, it downloads a small helper program (a *driver*) into the PDP-11's memory through the console, starts it, and then uses it to read or write disk blocks over the same serial line.

Because the driver is loaded fresh each time and never written to disk, **pdp11xfer** can be used on a machine with no operating system at all, or one whose operating system cannot otherwise talk to the host – all it needs is a working console (ODT, a boot ROM, or a simulator console) and a serial line.

Typical uses include:

- Reading the contents of an RL01/RL02, RK05, RK06/RK07, RX01, RX02, or MSCP-attached disk into a disk-image file on the host, for backup or archival purposes.
- Writing a disk-image file from the host onto one of these drives, for example to restore a backup or to install software.
- Verifying that a serial link to a PDP-11 (real or simulated) is working correctly, independent of any particular disk controller.

2 Requirements

To use **pdp11xfer** you will need:

- A Linux host with the `pdp11xfer` program installed.
- A serial connection to the target PDP-11. This can be:
 - a real serial port or USB-to-serial adapter, wired to the PDP-11's console serial line; or
 - a pseudo-terminal (pty) connected to a PDP-11 running under the SimH simulator (see Section 3.2).
- The PDP-11 sitting at a console prompt it can load a program from: standard console ODT, an M9301 or M9312 boot ROM console, or (when running under SimH) SimH's own `sim>` command prompt.
- If reading or writing a disk image, a supported disk controller and drive attached to the PDP-11 (see Section 4.5).

3 Connecting to the PDP-11

3.1 Real hardware

Connect a serial cable (directly, or through a USB-to-serial adapter) between the host and the PDP-11's console serial line. Note the device path Linux assigns to it (for example `/dev/ttyUSB0` or `/dev/ttyS0`) – this is what you will pass to `pdp11xfer` with the `--port` option.

The baud rate given to `pdp11xfer` with `--baud` must match the baud rate the PDP-11's console is actually running at. The default is 9600. Supported rates are:

50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400,
4800, 9600, 19200, 38400, 57600, 115200, 230400

The serial line is always used at 8 data bits, no parity, 1 stop bit (8N1), which is the standard configuration for a PDP-11 console.

3.2 SimH-simulated PDP-11s

`pdp11xfer` can also talk to a PDP-11 simulated by SimH, as long as its simulated console is reachable as a byte stream from the Linux host. A common way to arrange this is to have SimH listen on a raw console TCP port (`set console telnet=PORT; notelnet` in the SimH configuration) and bridge that TCP port to a local pseudo-terminal with `socat`, for example:

```
socat -d -d PTY,raw,echo=0,link=/tmp/pdp11pty TCP:localhost:PORT
```

`pdp11xfer` is then pointed at `/tmp/pdp11pty` with `--port`. The `; notelnet` suffix is required: without it, SimH applies telnet protocol processing to the console stream, which corrupts the driver download partway through. A complete worked example, including a SimH configuration file, is given in Section 5.3.

Since a pty has no real UART and no risk of characters being lost to transmission timing, you may pass `--baud=0` in this case. A baud rate of 0 disables the artificial per-character pacing that real hardware needs, so transfers to a simulator run at full local speed. **Do not** use `--baud=0` on a real serial line – without pacing, the PDP-11 driver's receive loop will lose characters sent faster than it can consume them.

When the boot ROM console is itself running under SimH (loader type `m9312-simh`, see Section 4.4), SimH should be running the M9312 console, sitting at its `@` prompt, when `pdp11xfer` starts. If SimH has instead dropped back to its own `sim>` prompt on the same connection `pdp11xfer` is using, the ROM is restarted automatically.

4 Command-Line Reference

4.1 Synopsis

```
pdp11xfer [options]
```

Exactly one *mode* option must be given, selecting what `pdp11xfer` should do.

4.2 Modes

Option	Action
<code>-r, --read</code>	Read a disk image from the PDP-11 into the file given by <code>--image</code> .
<code>-w, --write</code>	Write the disk image given by <code>--image</code> onto the PDP-11's drive.
<code>-t, --test[=N]</code>	Run a protocol self-test: send <i>N</i> words (default 1024) to the driver and verify it echoes them back correctly. The driver must already be loaded; this mode never loads one.
<code>-L, --load-driver</code>	Load and start the controller's driver, then confirm that it responds; do not read or write anything. The driver is always loaded, even if one is already running.
<code>-P, --ping</code>	Check whether a driver is loaded and responding. Never loads a driver, and needs no options other than <code>--port</code> and <code>--baud</code> .

4.3 Connection options

Option	Meaning
<code>-p, --port=PATH</code>	Serial device or pty path (required).
<code>-b, --baud=N</code>	Baud rate (default 9600; see Section 3.2 for 0).
<code>--no-rle</code>	Disable RLE compression of the transfer stream.

The `--no-rle` option is a diagnostic aid: if you suspect the run-length-encoded transfer format is causing a problem on a particular link, disabling it sends every word uncompressed, at some cost in transfer speed.

4.4 Driver loading options

Option	Meaning
<code>--loader=TYPE</code>	Console type to use when loading the driver (see below).

TYPE is one of:

Value	Console
<code>odt16, odt18, odt22</code>	Standard PDP-11 console ODT (the three values are equivalent; they refer to the address width of the front-panel/console display and do not change loader behavior).
<code>m9312</code>	M9312 boot ROM console (prompt @).
<code>m9301</code>	M9301 boot ROM console (prompt \$).
<code>m9312-simh</code>	An M9312 boot ROM console running under SimH. Behaves like <code>m9312</code> , but also detects and recovers from SimH having dropped the simulated CPU back to its own <code>sim></code> prompt, restarting the ROM automatically.
<code>simh</code>	SimH's own <code>sim></code> command console (not a PDP-11 console at all – see Section 3.2).

--loader is required for --load-driver. For --read/--write it is optional: if a driver is already loaded and responding, **pdp11xfer** uses it as-is and --loader is not needed; it is only required when a driver still needs to be loaded.

4.5 Device options

Option	Meaning
--controller= <i>NAME</i>	Disk controller type (required for --read/--write and for --load-driver).
--device= <i>NAME</i>	Drive type (required for --read/--write).
--unit= <i>N</i>	Drive unit: 0 (default) or 1, for a controller with more than one drive attached.
-i, --image= <i>PATH</i>	Disk image file (required for --read/--write).

Supported --controller/--device combinations, and the resulting disk-image size, are:

Controller	Device	Geometry (cyl. × head × sect.)	Image size
RL11	RL01	256 × 2 × 40, 256 B/block	5,242,880 bytes
RL11	RL02	512 × 2 × 40, 256 B/block	10,485,760 bytes
RK11	RK05	203 × 2 × 12, 512 B/block	2,494,464 bytes
RK611	RK06	411 × 3 × 22, 512 B/block	13,888,512 bytes
RK611	RK07	815 × 3 × 22, 512 B/block	27,540,480 bytes
RX11	RX01	77 × 1 × 26, 128 B/block	256,256 bytes
RX211	RX02_SD	77 × 1 × 26, 128 B/block (single density)	256,256 bytes
RX211	RX02_DD	77 × 1 × 26, 256 B/block (double density)	512,512 bytes
MSCP	MSCP	discovered automatically from the controller	varies by unit

The MSCP controller represents any MSCP-protocol storage controller (e.g. a UDA50 or an RQDX1/2/3), and MSCP is the single generic device type for it. Unlike the other controllers, MSCP drives are not identified by a fixed geometry table: **pdp11xfer** queries the controller for the attached unit's media type, block count, and block size once the driver is running, and reports what it found.

4.6 Other options

Option	Meaning
--progress= <i>STYLE</i>	Progress bar style: auto (default), full, basic, ascii, or off; see Section 5.1.
-V, --verbose	Print detailed protocol and driver loading progress to standard error.
-v, --version	Print the program version and exit.
-h, --help	Print a usage summary and exit.

4.7 Exit status

pdp11xfer exits with status 0 on success, and 1 if an error occurred (a bad command line, a communications failure, a driver-reported error, or a local file error). Diagnostic messages are printed

to standard error; progress messages go to standard output.

5 Typical Workflow

A normal session looks like this:

1. Bring the PDP-11 to a console prompt (power it up, or halt it if it is already running something else), or, for SimH, make sure the simulator is available and at the appropriate prompt for the chosen loader type.
2. Decide which console type applies (Section 4.4) and which controller and drive are attached (Section 4.5).
3. Optionally, load the driver with `--load-driver`, which confirms that it responds, and check the link with `--ping` or `--test` before attempting a real transfer.
4. Run `--read` or `--write` with the appropriate `--controller`, `--device`, `--image`, and (if the driver is not already loaded) `--loader` options.

For `--read` and `--write`, **pdp11xfer** automatically checks whether a compatible driver is already loaded and responding, and only loads one if needed – so it is safe to run several read/write operations in a row against the same session without reloading the driver each time, as long as the PDP-11 has not been reset or halted in between. Before the first transfer of a session, it also resets the controller and drive (for MSCP, this doubles as the step that discovers the unit’s geometry).

When writing, an image file smaller than the drive writes only the whole blocks it contains, and one larger than the drive writes only as much as fits; **pdp11xfer** says which before the transfer starts, and warns if the file ends with a partial block, which is not written.

5.1 The progress display

While the driver is downloaded, and while a disk is read or written, **pdp11xfer** shows a one-line progress display if standard output is a terminal. Downloading the driver takes one console round trip per word, so it can take a while even at high baud rates; its bar counts the words deposited:

```
/ 52% [=====>.....] 404/774 words 95 words/s ETA 0:04
```

For a disk transfer the bar comes beneath a short legend. In its plain ASCII form it looks like this:

```
# data    = RLE-compressed    . remaining
/ 62% [=====###>.....] 156.0/250.2 KB 3.1 KB/s RLE saved 41% ETA 0:52
```

The bar stands for the whole transfer, left to right from the first block. Finished parts are drawn solid if they went over the line at more than half their uncompressed size, and textured if RLE compression cut them to less than half, which is typical of empty or zero-filled areas of a disk. The track being transferred is highlighted and advances as its data is sent or received, so the display keeps moving on slow links. After the bar come the amount transferred, the average rate, how much RLE has reduced the traffic on the line so far, and the estimated time remaining; the least important of these are dropped on a narrow terminal. The rate and time remaining change with how well the data compresses, so they are only estimates.

How the bar is drawn depends on what the terminal can show. The `--progress` option selects a style, and by default (auto) **pdp11xfer** picks one from the `TERM` environment variable and the locale:

full Unicode block characters, with a spinner at the left and a smooth leading edge drawn in eighths of a character. Chosen for modern terminal emulators (Kitty, Ghostty, GNOME Terminal, xterm, and so on) in a UTF-8 locale.

basic Only the full, shaded, and half block characters, which console fonts include, with an ASCII spinner and brackets. Chosen when `TERM` is `linux` (the Linux text console) or begins with `cons` (the BSD consoles).

ascii The plain ASCII form shown above. Chosen when the locale is not UTF-8, or `TERM` is `dumb` or a VT-series terminal (`vt100`, `vt220`, and so on).

off No bar; only the summary line.

Color uses only the eight basic ANSI colors and dim text, and is turned off when the `NO_COLOR` environment variable is set or `TERM` is `dumb` or a VT-series terminal. If the bar looks wrong on a particular terminal, `--progress=ascii` will work on anything. With `--verbose`, disk transfers print one plain line per track instead, since the verbose log would break up a redrawn line, and the driver download shows no bar (the log already shows each deposit). If standard output is not a terminal, no bar is drawn. In every case a summary line is printed when a disk transfer completes, for example:

```
Wrote 250.2 KB (2002 blocks) in 0:44, 5.7 KB/s; RLE cut line traffic by 88%.
```

5.2 Examples

Run a protocol self-test against a driver that is already loaded:

```
pdp11xfer -p /dev/ttyUSB0 -b 9600 -t
```

Check whether a driver is already loaded and responding, without loading one:

```
pdp11xfer -p /dev/ttyUSB0 -b 9600 -P
```

Load the RL11 driver via console ODT, and confirm it responds:

```
pdp11xfer -p /dev/ttyUSB0 -b 9600 --loader=odt22 --controller=RL11 -L
```

Load the driver, then write `myimage.dsk` to an RL02 on an RL11:

```
pdp11xfer -p /dev/ttyUSB0 -b 9600 --loader=odt22 \
--controller=RL11 --device=RL02 -w -i myimage.dsk
```

Read the contents of an RK07 on an RK611 into a file, with verbose logging, over a SimH pty link:

```
pdp11xfer -p /tmp/pdp11pty -b 0 --loader=m9312-simh \
--controller=RK611 --device=RK07 -r -i rk07backup.dsk -V
```

5.3 A complete SimH example

This example sets up a simulated PDP-11/34 with an M9312 console ROM and an RX01 floppy drive, writes a disk image from the host onto the simulated drive, and then reads it back. It needs

the SimH `pdp11` simulator and `socat`, plus two ROM image files that are included in the `tests` directory of the `pdp11xfer` distribution:

File	Contents
<code>23-248F1.bin</code>	The M9312 console/diagnostic ROM, which provides the @ console prompt the driver is loaded through.
<code>empty-boot-prom.bin</code>	An empty ROM, used to fill the M9312's boot ROM sockets.

The SimH configuration file

Copy the two ROM files into an empty working directory, and create a file named `pdp11.ini` there containing:

```
; PDP-11/34 with an M9312 console ROM and an RX01 floppy drive,
; with its console on raw TCP port 2311 for use by pdp11xfer.
set console pchar=1000023600
set console telnet=2311;notelnet
set cpu 11/34
set cpu 256K

; M9312 console/diagnostic ROM, plus empty boot ROM sockets
set rom0 address=17765000
attach rom0 23-248F1.bin
set rom1 address=17773000
attach rom1 empty-boot-prom.bin
set rom2 address=17773200
attach rom2 empty-boot-prom.bin
set rom3 address=17773400
attach rom3 empty-boot-prom.bin

; the drive to transfer to and from
attach rx0 rx01.dsk

; start the M9312 console, which prints its '@' prompt
run 165144
```

The ROM addresses are given in SimH's 22-bit form: 17765000 is the M9312 console ROM's usual address of 165000, and 165144 is its console entry point. A real M9312 has four optional boot ROM sockets, at 17773000, 17773200, 17773400, and 17773600, in addition to the console ROM. SimH provides only four ROM slots in all, so after the console ROM only the first three boot sockets can be filled, and the socket at 17773600 is left empty.

The empty ROM image in the three boot sockets is needed because of how the M9312 console code looks for boot ROMs installed in the M9312 board. When the user types a device mnemonic at the @ prompt, or just presses Return, the console scans each boot ROM socket in turn and checks the first word there. An empty socket on a real M9312 reads as 177777, which the console takes to mean that no ROM is present, so it moves on to the next socket. A SimH ROM slot with nothing attached to it does not read that way, so the console decides that a ROM is present

and jumps into it, where it finds 000000 (a HALT instruction) and halts the simulated processor. `empty-boot-prom.bin` has 177777 as its first word, just as an empty socket would, so with it attached to the unused sockets the console skips over them. Typing Return, or the mnemonic of a device that has no boot ROM installed, then leaves the console running instead of halting the simulator.

The `attach rx0` line names the file SimH uses as the contents of the simulated RX01 diskette; if it does not exist, SimH creates it. To use a different controller, replace that line with the SimH commands for the drive you want (for example `set r10 r102` and `attach r10 r102.dsk` for an RL02 on an RL11), and change `--controller` and `--device` below to match.

Starting the simulator

In one terminal window, start SimH from the working directory:

```
pdp11 pdp11.ini
```

SimH prints

```
Listening on port 2311
```

```
...
```

```
Waiting for console Telnet connection
```

and waits. In a second terminal window, connect that port to a pseudo-terminal:

```
socat PTY,raw,echo=0,link=/tmp/pdp11pty TCP:localhost:2311
```

SimH prints `Running`, and the simulated M9312 console sends its `@` prompt to the pseudo-terminal. Leave both programs running.

Transferring an image

In a third terminal window, write the 256,256-byte image file `floppy.dsk` onto the simulated RX01:

```
pdp11xfer -p /tmp/pdp11pty -b 0 --loader=m9312-simh \  
--controller=RX11 --device=RX01 -w -i floppy.dsk
```

pdp11xfer finds no driver running, loads the RX11 driver through the `@` prompt, and writes the image:

```
pdp11xfer 1.0
```

```
Copyright (c) 2026 Large Scale Systems Museum  
By Dave McGuire <mcguire@lssmuseum.org>
```

```
No driver detected; loading now.  
Loading RX11 driver via M9312 (SimH) (774 words)...  
Starting driver at 010000...  
Driver loaded and started.  
Resetting UNIBUS and initializing controller...  
Controller initialized.  
Writing RX01 from floppy.dsk (2002 x 128-byte blocks, 256256 bytes)...  
Wrote 250.2 KB (2002 blocks) in 0:44, 5.7 KB/s; RLE cut line traffic by 88%.
```

(Progress bars, described in Section 5.1, are also shown while the driver is downloaded and while the blocks are transferred. The time, rate, and RLE figures depend on the image's contents.) To read the diskette back into a new file:

```
pdp11xfer -p /tmp/pdp11pty -b 0 \  
--controller=RX11 --device=RX01 -r -i readback.dsk
```

No `--loader` is needed this time, since the driver from the first command is still running; **pdp11xfer** reports `Driver already loaded and responding; skipping load.` and goes straight to the transfer. The two files can then be compared with `cmp floppy.dsk readback.dsk`.

Shutting down

When finished, type Ctrl-E in SimH's window to stop the simulation, and then quit at its `sim>` prompt. Stop socat with Ctrl-C. SimH keeps some disk images, including RX01 diskettes, in memory while it runs, and writes them out to their files only when it exits (it reports `RX0: writing buffer to file: rx01.dsk`), so be sure to quit SimH before using `rx01.dsk` itself. After SimH is restarted, the driver must be loaded again with `--loader`.

6 The PDP-11-Side Driver Programs

To do anything beyond talking to the console, **pdp11xfer** needs a small program running on the PDP-11 that can access the disk controller directly and exchange data over the console serial line. This program – referred to throughout this manual as *the driver*¹ – is what `--load-driver` (and, implicitly, `--read/--write`) deposits into memory and starts.

The driver is small (well under a page of PDP-11 memory), is loaded fresh into low memory at the start of each session, and is never written to the disk it is transferring. It performs no filesystem-level interpretation of the disk's contents: it only knows how to move raw blocks (sectors) between the drive and memory, and to exchange those blocks with the host over the serial line using a compact, error-checked encoding. All the decisions about which blocks make up a useful disk image are made on the host side.

Every controller's driver shares the same basic structure and the same simple command set (reset the controller, initialize/select the drive, read a block range, write a block range, plus a loop-back self-test used by `--test` and `--ping`), adapted to that controller's specific register layout:

RL11 Drives RL01 and RL02 cartridge disks attached to an RL11 (or compatible RLV11/RLV12) controller, addressing them by cylinder, head, and sector.

RK11 Drives RK05 disk cartridges attached to an RK11 controller, addressing them by cylinder, head, and sector.

RK611 Drives the larger RK06 and RK07 disk cartridges attached to an RK611 controller. Unlike the smaller drives, an RK06/RK07 can seek independently while a transfer to another part of the disk is in progress, but this driver transfers one track at a time regardless.

¹The PDP-11-side driver programs, and the serial transfer protocol they implement, were developed by Jörg Hoppe as part of his PDP11GUI package.

RX11 Drives RX01 8-inch floppy diskettes attached to an RX11 controller.

RX211 Drives RX02 floppy diskettes, in single or double density, attached to an RX211 (or compatible RXV21) controller, which adds DMA transfers and double-density support over the older RX11. (Use `--device=RX02_SD` for a single-density diskette; RX01 is not accepted on this controller.)

MSCP Drives any disk unit presented through the MSCP (Mass Storage Control Protocol) command set, as implemented by controllers such as the UDA50 or the RQDX1/2/3. Because MSCP controllers report their own unit geometry and capacity, this driver includes the extra initialization exchange needed to read that information back to the host, rather than relying on a fixed table.

Since the driver lives only in PDP-11 memory for the duration of a session, halting or resetting the CPU (other than the loader's own start command) will require it to be reloaded before the next operation.

7 Troubleshooting

7.1 No driver detected

```
error: no driver detected, and --loader was not given, so it
cannot be loaded.
```

This means `--read` or `--write` could not find a driver already running, and no `--loader` was given so one could not be loaded automatically. Add `--loader=TYPE` matching the PDP-11's current console (Section 4.4).

7.2 Controller/device mismatch

```
error: CONTROLLER does not support DEVICE
```

The `--controller` and `--device` you gave are not a valid combination – see the table in Section 4.5.

7.3 Timeouts and communication errors

If the link to the PDP-11 is not working correctly, operations will time out or report checksum/protocol errors rather than hanging forever. In these cases:

- Re-check that `--baud` matches the console's actual speed (Section 3.2 covers the `pty/SimH` exception).
- Run with `--ping` first to confirm the driver is present and answering, independent of any disk operation.
- Run with `--verbose` to see the detailed exchange being attempted.
- Confirm the PDP-11 was actually at the expected console prompt (ODT, M9312, M9301, or SimH's `sim>`) when **pdp11xfer** started loading the driver – if it was running something else,

or already running the driver from an earlier session, loading will fail or behave unexpectedly.

If a `--read` or `--write` fails, **pdp11xfer** first reports which part failed:

Message	Meaning
local file I/O error	The disk-image file on the host could not be read or written (check its path, permissions, and free space).
transfer error	Communication with the driver failed; the next line gives one of the communication messages below.
driver reported an error	The driver ran the operation but reported a failure (see below).

The communication messages **pdp11xfer** can report break down as follows:

Message	Meaning
receive timeout	No (or an incomplete) response was received in time.
illegal character received	An unexpected byte appeared in the response stream.
checksum error	The response failed its integrity check.
RLE repeat overflow	A malformed compressed response was received.
buffer overflow	The response was larger than the internal buffer allows.
driver reported an error	The driver ran the operation but reported a failure code (e.g. a real disk error).
serial I/O error	The operating system reported an error reading or writing the serial device itself.
aborted	The operation was cancelled before completing.

For a driver reported an error result during `--read/--write`, **pdp11xfer** also prints the driver's numeric error code and an information word, which reflect the disk controller's own error status; consult the controller's technical manual to interpret them.

7.4 Loader-specific errors

Message	Meaning
no prompt received (timeout)	The expected console prompt (@, \$, or <code>sim></code>) never appeared. Check that the PDP-11 is actually at that console and that <code>--loader</code> matches it.
serial I/O error	As above, an operating-system-level I/O error.
aborted	The load was cancelled before completing.