Setting Up Unix – Seventh Edition

Charles B. Haley
Dennis M. Ritchie

The distribution tape can be used only on a DEC PDP11/45 or PDP11/70 with RP03, RP04, RP05, RP06 disks and with a TU10, TU16, or TE16 tape drive. It consists of some preliminary bootstrapping programs followed by two file system images; if needed, after the initial construction of the file systems individual files can be extracted. (See restor(1))

If you are set up to do it, it might be a good idea immediately to make a copy of the tape to guard against disaster. The tape is 9-track 800 BPI and contains some 512-byte records followed by many 10240-byte records. There are interspersed tapemarks.

The system as distributed contains binary images of the system and all the user level programs, along with source and manual sections for them—about 2100 files altogether. The binary images, along with other things needed to flesh out the file system enough so UNIX will run, are to be put on one file system called the 'root file system'. The file system size required is about 5000 blocks. The file second system has all of the source and documentation. Altogether it amounts to more than 18,000 512-byte blocks.

Making a Disk From Tape

Perform the following bootstrap procedure to obtain a disk with a root file system on it.

1. Mount the magtape on drive 0 at load point.
2. Mount a formatted disk pack on drive 0.
3. Key in and execute at 100000

TU10	TU16/TE16
012700	Use the DEC ROM or other
172526	means to load block 1
010040	(i.e. second block) at 800 BPI
012740	into location 0 and transfer
060003	to 0.
000777	

   The tape should move and the CPU loop. (The TU10 code is *not* the DEC bulk ROM for tape; it reads block 0, not block 1.)
4. If you used the above TU10 code, halt and restart the CPU at 0, otherwise continue to the next step.
5. The console should type

   Boot
   :

   Copy the magtape to disk by the following procedure. The machine's printouts are shown in italic, explanatory comments are within ( ). Terminate each line you type by carriage return or line-feed. There are two classes of tape drives: the name 'tm' is used for the TU10, and 'ht' is used for the TU16 or TE16. There are also two classes of disks: 'rp' is used for the RP03, and 'hp' is used for the RP04/5/6.

If you should make a mistake while typing, the character '#' erases the last character typed up to the beginning of the line, and the character '@' erases the entire line typed. Some consoles cannot print lower case letters, adjust the instructions accordingly.

(bring in the program mkfs)
: tm(0,3)   (use ‘ht(0,3)’ for the TU16/TE16)
file system size: 5000
file system: rp(0,0)   (use ‘hp(0,0)’ for RP04/5/6)
isize = XX
m/n = XX
(after a while)
exit called
Boot
:

This step makes an empty file system.

6. The next thing to do is to restore the data onto the new empty file system. To do this you respond to the ‘*:*’ printed in the last step with

    (bring in the program restor)
    : tm(0,4)   (‘ht(0,4)’ for TU16/TE16)
    tape? tm(0,5)   (use ‘ht(0,5)’ for TU16/TE16)
    disk? rp(0,0)   (use ‘hp(0,0)’ for RP04/5/6)
    Last chance before scribbling on disk. (you type return)
    (the tape moves, perhaps 5-10 minutes pass)
    end of tape
    Boot
    :

    You now have a UNIX root file system.

Booting UNIX

You probably have the bootstrap running, left over from the last step above; if not, repeat the boot process (step 3) again. Then use one of the following:

: rp(0,0)rptmunix   (for RP03 and TU10)
: rp(0,0)rphtunix   (for RP03 and TU16/TE16)
: hp(0,0)hptmunix   (for RP04/5/6 and TU10)
: hp(0,0)hphtunix   (for RP04/5/6 and TU16/TE16)

The machine should type the following:

mem = xxx
#

The *mem* message gives the memory available to user programs in bytes.

UNIX is now running, and the ‘UNIX Programmer’s manual’ applies; references below of the form X(Y) mean the subsection named X in section Y of the manual. The ‘#’ is the prompt from the Shell, and indicates you are the super-user. The user name of the super-user is ‘root’ if you should find yourself in multi-user mode and need to log in; the password is also ‘root’.

To simplify your life later, rename the appropriate version of the system as specified above plain ‘unix.’ For example, use mv (1) as follows if you have an RP04/5/6 and a TU16 tape:

mv hphtunix unix

In the future, when you reboot, you can type just

hp(0,0)unix

to the ‘:’ prompt. (Choose appropriately among ‘hp’, ‘rp’, ‘ht’, ‘tm’ according to your configuration).

You now need to make some special file entries in the dev directory. These specify what sort of disk you are running on, what sort of tape drive you have, and where the file systems are. For simplicity, this recipe creates fixed device names. These names will be used below, and some of them are built into various programs, so they are most convenient. However, the names do not always represent the actual major and minor device in the manner suggested in section 4 of the Programmer's Manual. For example, 'rp3' will be used for the name of the file system on which the user file system is put, even though it might be on an RP06 and is not logical device 3. Also, this sequence will put the user file system on the same disk drive as the root, which is not the best place if you have more than one drive. Thus the prescription below should be taken only as one example of where to put things. See also the section on 'Disk layout' below.

In any event, change to the dev directory (cd(1)) and, if you like, examine and perhaps change the makefile there (make (1)).

```
cd /dev
cat makefile
```

Then, use one of

```
make rp03
make rp04
make rp05
make rp06
```

depending on which disk you have. Then, use one of

```
make tm
make ht
```

depending on which tape you have. The file 'rp0' refers to the root file system; 'swap' to the swap-space file system; 'rp3' to the user file system. The devices 'rrp0' and 'rrp3' are the 'raw' versions of the disks. Also, 'mt0' is tape drive 0, at 800 BPI; 'rmt0' is the raw tape, on which large records can be read and written; 'nrmt0' is raw tape with the quirk that it does not rewind on close, which is a subterfuge that permits multifile tapes to be handled.

The next thing to do is to extract the rest of the data from the tape. Comments are enclosed in ( ); don't type these. The number in the first command is the size of the file system; it differs between RP03, RP04/5, and RP06.

```
/etc/mkfs /dev/rp3 74000          (153406 if on RP04/5, 322278 on RP06)
(The above command takes about 2-3 minutes on an RP03)
dd if=/dev/nrmt0 of=/dev/null bs=20b files=6        (skip 6 files on the tape)
restor rf /dev/rmt0 /dev/rp3       (restore the file system)
(Reply with a 'return' (CR) to the 'Last chance' message)
(The restor takes about 20-30 minutes)
```

All of the data on the tape has been extracted.

You may at this point mount the source file system (mount(1)). To do this type the following:

```
/etc/mount /dev/rp3 /usr
```

The source and manual pages are now available in subdirectories of /usr.

The above mount command is only needed if you intend to play around with source on a single user system, which you are going to do next. The file system is mounted automatically when multi-user mode is entered, by a command in the file /etc/rc. (See 'Disk Layout' below).

Before anything further is done the bootstrap block on the disk (block 0) should be filled in. This is done using the command

```
dd if=/usr/mdec/rpuboot of=/dev/rp0 count=1
```

if you have the RP03, or

```
dd if=/usr/mdec/hpuboot of=/dev/rp0 count=1
```

if you have an RP04/5/6. Now the DEC disk bootstraps are usable. See Boot Procedures(8) for further information.

Before UNIX is turned up completely, a few configuration dependent exercises must be performed. At this point, it would be wise to read all of the manuals (especially ‘Regenerating System Software’) and to augment this reading with hand to hand combat.

Reconfiguration

The UNIX system running is configured to run with the given disk and tape, a console, and no other device. This is certainly not the correct configuration. You will have to correct the configuration table to reflect the true state of your machine.

It is wise at this point to know how to recompile the system. Print (cat(1)) the file /usr/sys/conf/makefile. This file is input to the program ‘make(1)’ which if invoked with ‘make all’ will recompile all of the system source and install it in the correct libraries.

The program mkconf(1) prepares files that describe a given configuration (See mkconf(1)). In the /usr/sys/conf directory, the four files *xy*conf were input to mkconf to produce the four versions of the system *xy*unix. Pick the appropriate one, and edit it to add lines describing your own configuration. (Remember the console typewriter is automatically included; don’t count it in the kl specification.) Then run mkconf; it will generate the files l.s (trap vectors) c.c (configuration table), and mch0.s. Take a careful look at l.s to make sure that all the devices that you have are assembled in the correct interrupt vectors. If your configuration is non-standard, you will have to modify l.s to fit your configuration.

There are certain magic numbers and configuration parameters imbedded in various device drivers that you may want to change. The device addresses of each device are defined in each driver. In case you have any non-standard device addresses, just change the address and recompile. (The device drivers are in the directory /usr/sys/dev.)

The DC11 driver is set to run 4 lines. This can be changed in dc.c.

The DH11 driver is set to handle 3 DH11’s with a full complement of 48 lines. If you have less, or more, you may want to edit dh.c.

The DN11 driver will handle 4 DN’s. Edit dn.c.

The DU11 driver can only handle a single DU. This cannot be easily changed.

The KL/DL driver is set up to run a single DL11-A, -B, or -C (the console) and no DL11-E’s. To change this, edit kl.c to have NKL11 reflect the total number of DL11-ABC’s and NDL11 to reflect the number of DL11-E’s. So far as the driver is concerned, the difference between the devices is their address.

All of the disk and tape drivers (rf.c, rk.c, rp.c, tm.c, tc.c, hp.c, ht.c) are set up to run 8 drives and should not need to be changed. The big disk drivers (rp.c and hp.c) have partition tables in them which you may want to experiment with.

After all the corrections have been made, use ‘make(1)’ to recompile the system (or recompile individually if you wish: use the makefile as a guide). If you compiled individually, say ‘make unix’ in the directory /usr/sys/conf. The final object file (unix) should be moved to the root, and then booted to try it out. It is best to name it /nunix so as not to destroy the working system until you’re sure it does work. See Boot Procedures(8) for a discussion of booting. Note: before taking the system down, always (!!) perform a sync(1) to force delayed output to the disk.

Special Files

Next you must put in special files for the new devices in the directory /dev using mknod(1). Print the configuration file c.c created above. This is the major device switch of each device class (block and character). There is one line for each device configured in your system and a null line for place holding for those devices not configured. The essential block special files were installed above; for any new devices, the major device number is selected by counting the line number (from zero) of the device’s entry in the block configuration table. Thus the first entry in the table bdevsw would be major device zero. This number is also printed in the table along the right margin.

The minor device is the drive number, unit number or partition as described under each device in section 4 of the manual. For tapes where the unit is dial selectable, a special file may be made for each possible selection. You can also add entries for other disk drives.

In reality, device names are arbitrary. It is usually convenient to have a system for deriving names, but it doesn't have to be the one presented above.

Some further notes on minor device numbers. The hp driver uses the 0100 bit of the minor device number to indicate whether or not to interleave a file system across more than one physical device. See hp(4) for more detail. The tm and ht drivers use the 0200 bit to indicate whether or not to rewind the tape when it is closed. The 0100 bit indicates the density of the tape on TU16 drives. By convention, tape special files with the 0200 bit on have an 'n' prepended to their name, as in /dev/nmt0 or /dev/nrmt1. Again, see tm(4) or ht(4).

The naming of character devices is similar to block devices. Here the names are even more arbitrary except that devices meant to be used for teletype access should (to avoid confusion, no other reason) be named /dev/ttyX, where X is some string (as in '00' or 'library'). The files console, mem, kmem, and null are already correctly configured.

The disk and magtape drivers provide a 'raw' interface to the device which provides direct transmission between the user's core and the device and allows reading or writing large records. The raw device counts as a character device, and should have the name of the corresponding standard block special file with 'r' prepended. (The 'n' for no rewind tapes violates this rule.) Thus the raw magtape files would be called /dev/rmtX. These special files should be made.

When all the special files have been created, care should be taken to change the access modes (chmod(1)) on these files to appropriate values (probably 600 or 644).

Floating Point

UNIX only supports (and really expects to have) the FP11-B/C floating point unit. For machines without this hardware, there is a user subroutine available that will catch illegal instruction traps and interpret floating point operations. (See fptrap(3).) To install this subroutine in the library, change to /usr/src/libfpsim and execute the shell files

```
compall
mklib
```

The system as delivered does not have this code included in any command, although the operating system adapts automatically to the presence or absence of the FP11.

Next, a floating-point version of the C compiler in /usr/src/cmd/c should be compiled using the commands:

```
cd /usr/src/cmd/c
make fc1
mv fc1 /lib/fc1
```

This allows programs with floating point constants to be compiled. To compile floating point programs use the '−f' flag to cc(1). This flag ensures that the floating point interpreter is loaded with the program and that the floating point version of 'cc' is used.

Time Conversion

If your machine is not in the Eastern time zone, you must edit (ed(1)) the file /usr/sys/h/param.h to reflect your local time. The manifest 'TIMEZONE' should be changed to reflect the time difference between local time and GMT in minutes. For EST, this is 5*60; for PST it would be 8*60. Finally, there is a 'DSTFLAG' manifest; when it is 1 it causes the time to shift to Daylight Savings automatically between the last Sundays in April and October (or other algorithms in 1974 and 1975). Normally this will not have to be reset. When the needed changes are done, recompile and load the system using make(1) and install it. (As a general rule, when a system header file is changed, the entire system should be recompiled. As it happens, the only uses of these flags are in /usr/sys/sys/sys4.c, so if this is all that was changed it alone needs to be recompiled.)

You may also want to look at timezone(3) (/usr/src/libc/gen/timezone.c) to see if the name of your timezone is in its internal table. If needed, edit the changes in. After timezone.c has been edited it should be compiled and installed in its library. (See /usr/src/libc/(mklib and compall)) Then you should (at your leisure) recompile and reinstall all programs that use it (such as date(1)).

Disk Layout

If there are to be more file systems mounted than just the root and /usr, use mkfs(1) to create any new file system and put its mounting in the file /etc/rc (see init(8) and mount(1)). (You might look at /etc/rc anyway to see what has been provided for you.)

There are two considerations in deciding how to adjust the arrangement of things on your disks: the most important is making sure there is adequate space for what is required; secondarily, throughput should be maximized. Swap space is a critical parameter. The system as distributed has 8778 (hpunix) or 2000 (rpunix) blocks for swap space. This should be large enough so running out of swap space never occurs. You may want to change these if local wisdom indicates otherwise.

The system as distributed has all of the binaries in /bin. Most of them should be moved to /usr/bin, leaving only the ones required for system maintenance (such as icheck, dcheck, cc, ed, restor, etc.) and the most heavily used in /bin. This will speed things up a bit if you have only one disk, and also free up space on the root file system for temporary files. (See below).

Many common system programs (C, the editor, the assembler etc.) create intermediate files in the /tmp directory, so the file system where this is stored also should be made large enough to accommodate most high-water marks. If you leave the root file system as distributed (except as discussed above) there should be no problem. All the programs that create files in /tmp take care to delete them, but most are not immune to events like being hung up upon, and can leave dregs. The directory should be examined every so often and the old files deleted.

Exhaustion of user-file space is certain to occur now and then; the only mechanisms for controlling this phenomenon are occasional use of du(1), df(1), quot(1), threatening messages of the day, and personal letters.

The efficiency with which UNIX is able to use the CPU is largely dictated by the configuration of disk controllers. For general time-sharing applications, the best strategy is to try to split user files, the root directory (including the /tmp directory) and the swap area among three controllers.

Once you have decided how to make best use of your hardware, the question is how to initialize it. If you have the equipment, the best way to move a file system is to dump it (dump(1)) to magtape, use mkfs(1) to create the new file system, and restore (restor(1)) the tape. If for some reason you don't want to use magtape, dump accepts an argument telling where to put the dump; you might use another disk. Sometimes a file system has to be increased in logical size without copying. The super-block of the device has a word giving the highest address which can be allocated. For relatively small increases, this word can be patched using the debugger (adb(1)) and the free list reconstructed using icheck(1). The size should not be increased very greatly by this technique, however, since although the allocatable space will increase the maximum number of files will not (that is, the i-list size can't be changed). Read and understand the description given in file system(5) before playing around in this way. You may want to see section rp(4) for some suggestions on how to lay out the information on RP disks.

If you have to merge a file system into another, existing one, the best bet is to use tar(1). If you must shrink a file system, the best bet is to dump the original and restor it onto the new filesystem. However, this might not work if the i-list on the smaller filesystem is smaller than the maximum allocated inode on the larger. If this is the case, reconstruct the filesystem from scratch on another filesystem (perhaps using tar(1)) and then dump it. If you are playing with the root file system and only have one drive the procedure is more complicated. What you do is the following:

1. GET A SECOND PACK!!!!
2. Dump the current root filesystem (or the reconstructed one) using dump(1).

3. Bring the system down and mount the new pack.
4. Retrieve the WECo distribution tape and perform steps 1 through 5 at the beginning of this document, substituting the desired file system size instead of 5000 when asked for 'file system size'.
5. Perform step 6 above up to the point where the 'tape' question is asked. At this point mount the tape you made just a few minutes ago. Continue with step 6 above substituting a 0 (zero) for the 5.

New Users

Install new users by editing the password file /etc/passwd (passwd(5)). This procedure should be done once multi-user mode is entered (see init(8)). You'll have to make a current directory for each new user and change its owner to the newly installed name. Login as each user to make sure the password file is correctly edited. For example:

```
ed /etc/passwd
$a
joe::10:1::/usr/joe:
w
q
mkdir /usr/joe
chown joe /usr/joe
login joe
ls −la
login root
```

This will make a new login entry for joe, who should be encouraged to use passwd(1) to give himself a password. His default current directory is /usr/joe which has been created. The delivered password file has the user *bin* in it to be used as a prototype.

Multiple Users

If UNIX is to support simultaneous access from more than just the console terminal, the file /etc/ttys (ttys(5)) has to be edited. To add a new terminal be sure the device is configured and the special file exists, then set the first character of the appropriate line of /etc/ttys to 1 (or add a new line). Note that init.c will have to be recompiled if there are to be more than 100 terminals. Also note that if the special file is inaccessible when init tries to create a process for it, the system will thrash trying and retrying to open it.

File System Health

Periodically (say every day or so) and always after a crash, you should check all the file systems for consistency (icheck, dcheck(1)). It is quite important to execute sync (8) before rebooting or taking the machine down. This is done automatically every 30 seconds by the update program (8) when a multiple-user system is running, but you should do it anyway to make sure.

Dumping of the file system should be done regularly, since once the system is going it is very easy to become complacent. Complete and incremental dumps are easily done with dump(1). Dumping of files by name is best done by tar(1) but the number of files is somewhat limited. Finally if there are enough drives entire disks can be copied using cp(1), or preferably with dd(1) using the raw special files and an appropriate block size.

Converting Sixth Edition Filesystems

The best way to convert file systems from 6th edition (V6) to 7th edition (V7) format is to use tar(1). However, a special version of tar must be prepared to run on V6. The following steps will do this:

1. change directories to /usr/src/cmd/tar
2. At the shell prompt respond

   ```
   make v6tar
   ```

   This will leave an executable binary named 'v6tar'.

3. Mount a scratch tape.
4. Use tp(1) to put ‘v6tar’ on the scratch tape.
5. Bring down V7 and bring up V6.
6. Use tp (on V6) to read in ‘v6tar’. Put it in /bin or /usr/bin (or perhaps some other preferred location).
7. Use v6tar to make tapes of all that you wish to convert. You may want to read the manual section on tar(1) to see whether you want to use blocking or not. Try to avoid using full pathnames when making the tapes. This will simplify moving the hierarchy to some other place on V7 if desired. For example

   ```
   chdir /usr/ken
   v6tar c .
   ```

   is preferable to

   ```
   v6tar c /usr/ken
   ```

8. After all of the desired tapes are made, bring down V6 and reboot V7. Use tar(1) to read in the tapes just made.

Odds and Ends

The programs dump, icheck, quot, dcheck, ncheck, and df (source in /usr/source/cmd) should be changed to reflect your default mounted file system devices. Print the first few lines of these programs and the changes will be obvious. Tar should be changed to reflect your desired default tape drive.

Good Luck

Charles B. Haley
Dennis M. Ritchie