Getting Started with the Unix99 Operating System, Release 3

Table of Contents

Change History

9/7/2025	Original version
9/11/2025	Added steps for loading java packages for Unix99-specific TIPI unique features
9/14/2025	Added change history
	Added PicoPEB installation instructions
09/28/2025	Added trouble-shooting section
	Misc corrections
10/12/2025	Corrected Classic99 disk setup description
	Added version numbers for hardware with integrated software/firmware
	Added accounts management sections
	Added upgrade section
10/24/2025	Added example for loading a manual page
10/25/2025	Added table of observed boot times
12/17/2025	Added printer configuration

1. Purpose

This document describes the features, installation instructions and usage of Unix99 Release 3. Unix99 Release 3 provides a modern operating system environment for the TI-99/4A, combining the standards of Unix and the unique features of the TI-99/4A platform. This release includes an external development environment with C language cross-compiler and standard libraries. The development environment also includes extensions for the platform architecture.

1.1. Features

- Unix commands
- Unix APIs
- File streams (text and binary), block level read/write cached
- Directory handling
- Current working directory
- Path operators (/, ., .., and ~)
- Command line argument passing (argc, argv)
- Standard input/output with pipes and file redirection
- Process queuing via system(), execv()
- IPC (shared memory)
- Signal handling
- In-process cooperative multitasking
- User account management with password management
- Basic ciphers
- Login session management
- Manual pages
- Drive support (WDS, SCS, IDE, TIPI, DSK)
- Graphics I, Graphics II, Text (40 and 80 columns, 24 and 30 rows) display modes
- Drawing support
- Color management
- Font management
- Sprite management
- Text I/O management
- Sound management
- Speech management
- SAMS support
- File caching (executables only)
- Memory paging
- F18A / Pico9918 support
- GPU based text management
- GPU method management
- Dynamic link library, partially ROM-based
- ROM-based kernel
- TIPI support
- PicoPEB support
- Printer management
- External keyboard management
- Mouse support
- Joystick support
- Time management
- Basic GROFF support
- ROM paging

- Development pipeline
- C language cross compiler
- Type support for signed/unsigned char, int, long and double

1.2. Prerequisites

The Unix99 runtime environment can be used in an emulation environment (Classic99 and MAME) and on physical hardware.

Unix99 execution requires the following emulated or real hardware at a minimum:
- TI-99 console, providing CPU, graphics, VRAM and sound
- 32KB RAM
- At least one persistent storage device and minimum versions:
  - TIPI - software v2.6, DSR v2025-02-02
  - WDSx
  - IDEx - DSR v17, v18 preferred
  - SCSx - DSR v1.5 or v1.6
  - Classic99 on DSK1 - QI399.083
- Programmable GROM hardware, such as FinalGROM, that provides a host for the Unix99 ROM

Additional supported TI-99 hardware and minimum versions:
- Speech synthesizer
- PICO9918 - v1.0
- F18A - v1.9
- SAMS memory card
- RS232 and Parallel I/O
- External USB mouse (TIPI required)
- External USB keyboard (TIPI required)

The development environment requires:
- Linux or MacOS
- Docker Desktop

1.3. Performance

The following boot times have been observed, measuring time from the FinalGROM selection screen to the login prompt.

Configuration	**Boot time**	**Notes**
MAME		
WDS	34.4	MFM drive performance is emulated
IDE	16.0	IDE drive performance limited only by the host system
Classic99	16.7	Filesystem drive performance limited only by the host system
PEB		
TIPI	29.7	
IDE	26.2	
SCSI		Not measured
PPEB	18.5	

2. Installation

The Unix99 files required for installation are contained in the distribution file named unix99r3_<date>.tar.gz.

2.1. Extraction

Extract the Unix99 files from the unix99r3_<date>.tar.gz to your PC/Linux/Mac filesystem. The tar file includes a top-level directory named unix99r3. All other files and directories are contained therein.

The image directory contains the following directories and files, although the permissions, username, group, sizes and dates may differ. This is the runtime directory and is also used directly by Classic99.

```
drwxr-xr-x@ 52 marko  staff  1664 Aug 24 15:50 bin
-rw-r--r--@  1 marko  staff   384 Aug 25 18:06 bootconfig
drwxr-xr-x@  6 marko  staff   192 Jun 27 14:10 etc
drwxr-xr-x@ 37 marko  staff  1184 Aug 25 18:06 example
drwxr-xr-x@  4 marko  staff   128 Jun 27 14:10 fonts
drwxr-xr-x@  5 marko  staff   160 Jul 11 07:25 game
drwxr-xr-x@  5 marko  staff   160 Aug 25 18:14 home
drwxr-xr-x@  6 marko  staff   192 Jun 27 14:10 PLUGINS
drwxr-xr-x@  9 marko  staff   288 Aug 25 18:06 proc
drwxr-xr-x@  2 marko  staff    64 Aug 25 18:14 tmp
drwxr-xr-x@  3 marko  staff    96 Aug 11 11:20 usr
```

Other files, described for specific platform installation in subsequent sections, contain this directory structure.

The following subsections describe installation of the runtime in the supported environments.

2.2. Classic99

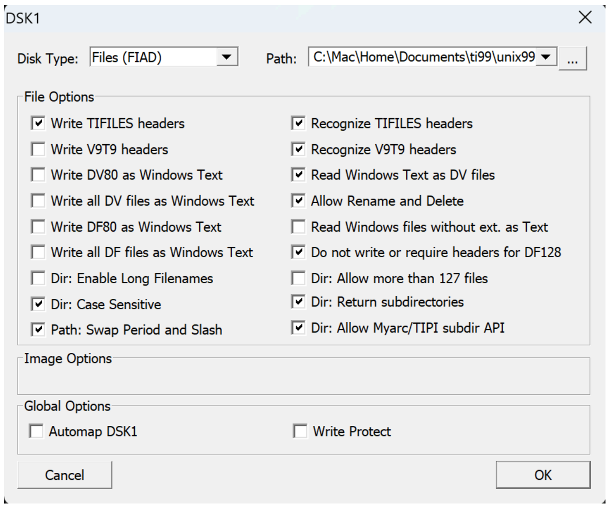
These instructions are applicable to Classic99 as of version QI399.083.

Classic99 will requires configuration of of one of its disk definitions. and a user cartridge.

These instructions use DSK1; however, other disks should be usable.

DSK1 should be configured to point to the unix99r3/image directory.
To configure DSK1, select the menu option “Disk->DSK1->.\DSK1\”.

The path to the unix99r3/image directory must be entered in the “Path:” field. Set all other configuration values as depicted in the following image.



DSK1

Disk Type: Files (FIAD)   Path: C:\Mac\Home\Documents\ti99\unix99

File Options

☑ Write TIFILES headers	☑ Recognize TIFILES headers
☐ Write V9T9 headers	☑ Recognize V9T9 headers
☐ Write DV80 as Windows Text	☑ Read Windows Text as DV files
☐ Write all DV files as Windows Text	☑ Allow Rename and Delete
☐ Write DF80 as Windows Text	☐ Read Windows files without ext. as Text
☐ Write all DF files as Windows Text	☑ Do not write or require headers for DF128
☐ Dir: Enable Long Filenames	☐ Dir: Allow more than 127 files
☑ Dir: Case Sensitive	☑ Dir: Return subdirectories
☑ Path: Swap Period and Slash	☑ Dir: Allow Myarc/TIPI subdir API

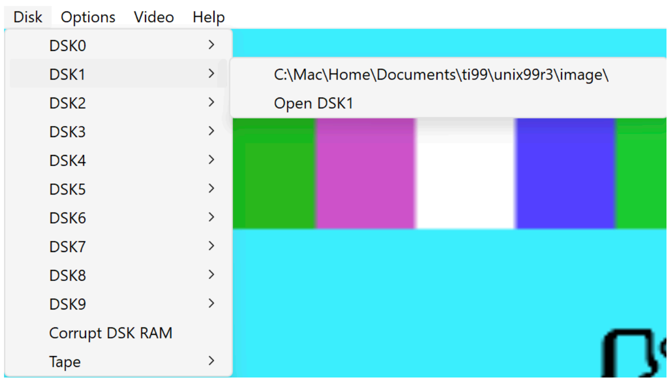
Image Options

Global Options

☐ Automap DSK1	☐ Write Protect

Press ‘OK’ and DSK1 is configured.

This picture shows the Classic99 Disk->DSK1 menu option with the full path to DSK1’s directory clearly visible.



C:\Mac\Home\Documents\ti99\unix99r3\image\

Next, the cartridge image must be set. Select the menu option “Cartridge->User->Open…”. Navigate to the unix99 directory, locate the file “uthree.bin”, and press “Open”.

Classic99 will perform a restart and be ready to run Unix99.

Follow the instructions below in the section ‘Startup’.

2.3. MAME

In MAME, Unix99 uses the Myarc WDS DSR at WDS1 for the root file system. An example script to run MAME, run_mame, is provided. As MAME can be and often is installed and configured differently by each user, it will be necessary to augment the run_mame script to correctly point to the MAME directories. In particular the environment variables MAME_PATH and MAME_BUILD_PATH must be adjusted.

Note that the run_mame script also removes state files created by MAME. If you desire the retention of state behavior the lines to remove are fairly obviously, beginning with the setting of the STFILE environment variable. My particular uses of MAME resulted in undesirable behavior.

The run_mame script stands up a fairly feature-packed TI-99/4a with a PEB, 1 MB SAMS, speech synthesizer, RS232/PIO, high density floppy drive controller, two floppy drives, and a Myarc WDS hard drive. Serial and parallel output are directed to the files serial.txt and parallel.txt.

The floppy drive images are floppy1.dsk and floppy2.dsk.
The hard drive image is unix99r3.hd.

When configuration is complete, the MAME emulator should start and provide the menu option "UNIX99R3".

Follow the instructions below in the section 'Startup'.

2.4. TI-99/4a with TIPI

The Unix99r3 kernel runs on a TI99 cartridge. FinalGROM is now the most commonly available and these instructions presume it's use. Other cartridge solutions should also work but have not been tested.

Copy the file "uthree.bin" to the FinalGROM's flash drive root directory.

Transfer the tar file "unix99r3.tar" to the TIPI using sftp:

```
$ sftp tipi@tipi.local                Update the tipi hostname as necessary
Enter the tipi host password
sftp> put unix99.tar
Uploading unix99r3.tar to /home/tipi/unix99r3.tar
sftp> exit
```

Login to the tipi host:
```
ssh tipi@tipi.local
Enter the tipi host password
```

Change directory to the tipi_disk path:
```
$ cd tipi_disk
$ pwd
/home/tipi/tipi_disk
```

Untar the file "unix99r3.tar" into the tipi_disk path:
```
$ tar xvpf ../unix99r3.tar
```

Remove any alternate stream files that were contained in the tar file:
```
$ find . -name "._*" -exec rm {} \;
```

The TIPI device provides a plug-in feature and Unix99r3 makes use of it to provide support for an external keyboard and mouse. These plugins and installation steps are required whether or not the user intends to use these features. TIPI plug-ins make use of the java evdev package.

Install the evdev package on the TIPI device:
```
$ cd ~/tipi/services
$ . ENV/bin/activate
$ pip install evdev
$ exit
```

TIPI should now be ready to run Unix99.

Follow the instructions below at ‘Startup’.

2.5. TI-99/4a with TIPI and any of IDE, SCS or WDS

Follow the complete set of instructions from above section “TI-99/4a with TIPI”.

Unix99r3 should have booted from TIPI.

For Unix99r3 to boot on any of IDE, SCS or WDS, it must be copied there. Unix99r3 locates its files by looking at the DSR root directories of TIPI, IDEx, SCSx and WDSx, searching for the first occurrence of the file named “bootconfig”. To be sure where Unix99 is booting from, ensure that only one occurrence of this file exists on the system.

Login with the username “test” and password “test”.

Execute the following commands:

```
$ cd /
$ cp -R bin bootconfig etc example /BOOTDEV          replace BOOTDEV with your device,
                                                     such as /SCS1
$ cp -R fonts game home proc /BOOTDEV
$ cp -R tmp usr /BOOTDEV
```

Remove the bootconfig file from the TIPI disk:

```
$ rm /bootconfig
```

Reboot:

```
$ reboot
```

After walking through the TI boot screens and selecting UNIX99R3, your system should be booted on the device you selected. After logging in, the df command can be used to identify the volume name of the booted disk. Use the command “df /“.

2.6. TI-99/4a with a combination of IDE, SCS or WDS

This is a more “advanced” and difficult configuration. The files in the tar file “unix99r3.tar” need to be copied into one of the paths IDE1-8, SCS1-8 or WDS1-8. Use your tool of choice to copy the directories and files under the directory “image” to the root directory of the desired volume.

Specific written instructions will gladly be included here if provided (hint hint).

2.7. TI-99/4a with Pico Peripheral Expansion Box (PEB)

Copy the contents of the “image” directory to the root directory of your PicoPEB SD card. Copy the cartridge file “uthree.bin” onto the SD card and rename to “uthree8.bin”. Set “CART=uthree” in the PicoPeb configuration file.

The PicoPEB should now be ready to run Unix99.

Follow the instructions below at ‘Startup’.

3. Upgrade from Previous Installation

It is recommended that the cartridge ROM and all executable files be replaced during the upgrade process, which is simply copying in new files.

Prior to copying, backup should include the following, if altered:

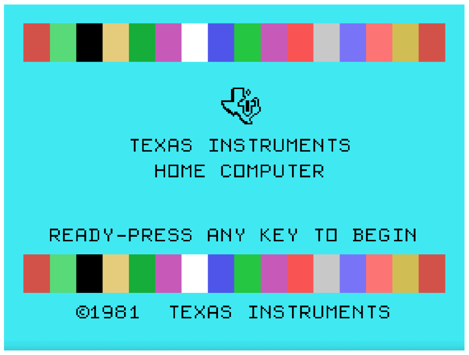
/etc/rc	if startup configurations were updated
/etc/passwd	if accounts were added or passwords changed
/home/*	if accounts were added or new files created

A future planned feature is an upgrade process.

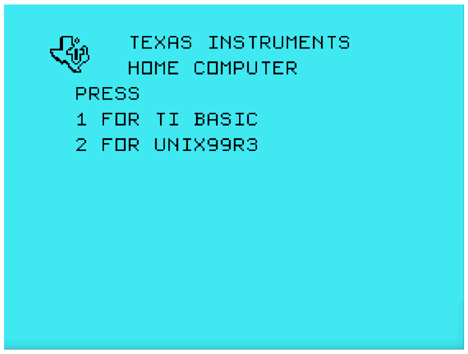
4. Usage

4.1. Power-Up

Power on the TI-99/4a computer and then select to run the UNIX99R3 program.

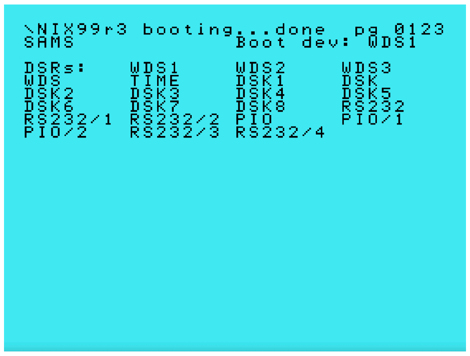


```
TEXAS INSTRUMENTS
HOME COMPUTER

READY-PRESS ANY KEY TO BEGIN

©1981   TEXAS INSTRUMENTS
```

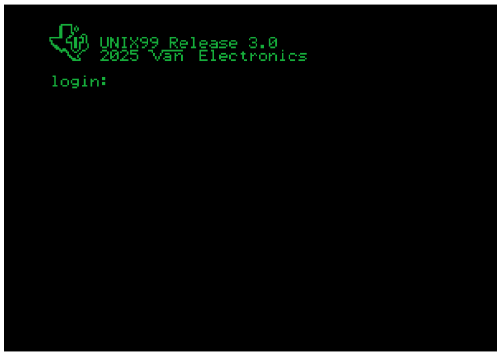


```
     TEXAS INSTRUMENTS
     HOME COMPUTER
PRESS
1 FOR TI BASIC
2 FOR UNIX99R3
```

It will start, initialize the environment, and present the login program.



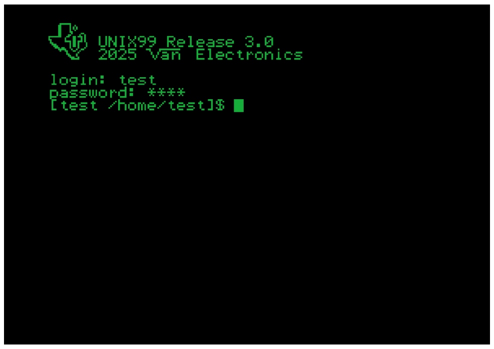
```
\NIX99r3 booting...done  pg 0123
SAMS                Boot dev: WDS1

DSRs:    WDS1     WDS2     WDS3
WDS      TIME     DSK1     DSK
DSK2     DSK3     DSK4     DSK5
DSK6     DSK7     DSK8     RS232
RS232/1  RS232/2  PIO      PIO/1
PIO/2    RS232/3  RS232/4
```



```
   UNIX99 Release 3.0
   2025 Van Electronics

login:
```

4.2. Login

Login using the username ‘test’ and password ‘test’.



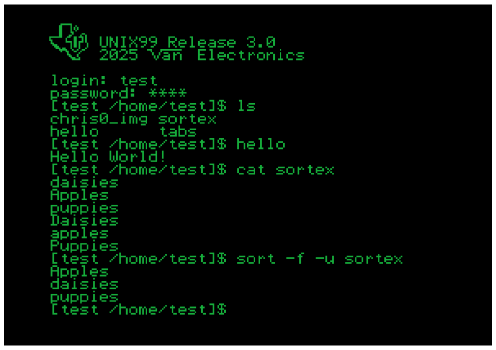
```
   UNIX99 Release 3.0
   2025 Van Electronics

login: test
password: ****
[test /home/test]$ █
```

After login, the shell, sh, will start. The shell is provided for command interpretation and execution.

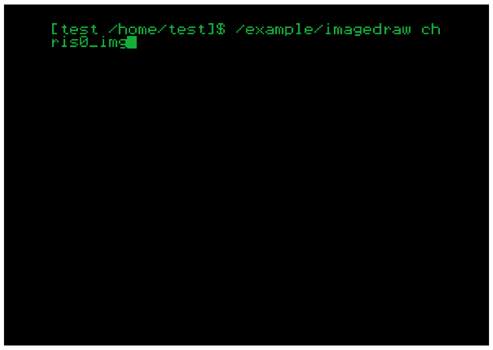
4.3. First Step Examples

Enter the following commands:

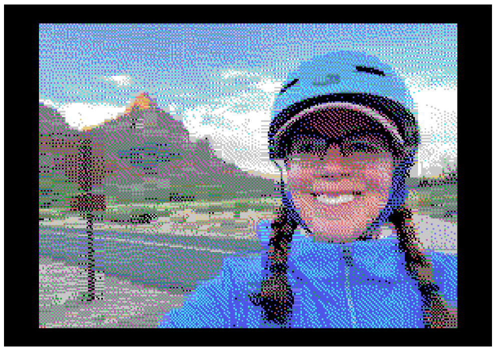
$ ls	lists the contents of the current working directory
$ hello	runs the program hello, which is the hello world example program
$ cat sortex	displays the contents of the file sortex, which is an unsorted file
$ sort -f -u sortex	sorts the file sortex, case insensitive and presenting only unique items
$ /example/imagedraw chris0_img	displays the image chris0_img
$ man vi	loads the manual page for the vi (visual) editor

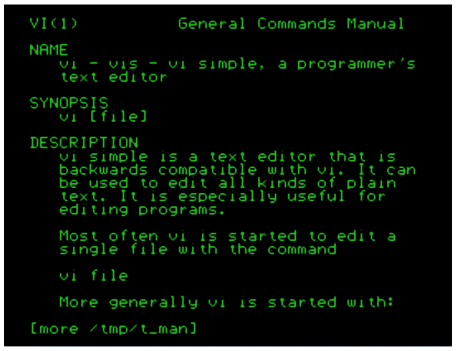


```
UNIX99 Release 3.0
2025 Van Electronics

login: test
password: ****
[test /home/test]$ ls
chris0_img sortex
hello      tabs
[test /home/test]$ hello
Hello World!
[test /home/test]$ cat sortex
daisies
Apples
puppies
Daisies
apples
Puppies
[test /home/test]$ sort -f -u sortex
Apples
daisies
puppies
[test /home/test]$
```



```
[test /home/test]$ /example/imagedraw ch
ris0_img
```





```
VI(1)            General Commands Manual

NAME
    vi - vis - vi simple, a programmer's
    text editor

SYNOPSIS
    vi [file]

DESCRIPTION
    vi simple is a text editor that is
    backwards compatible with vi. It can
    be used to edit all kinds of plain
    text. It is especially useful for
    editing programs.

    Most often vi is started to edit a
    single file with the command

    vi file

    More generally vi is started with:

[more /tmp/t_man]
```

4.4. Default User Accounts

The default user accounts are root, test and marko.

The test account contains a small number of files for the examples earlier in this section.

The marko account contains a larger number of files that are used for regression testing. There are no “personal” files there. The password is “password”.

The root account has no files but it’s use is required for a small number of system administration actions, requiring occasional use. The password is ‘password”.

4.5. User Accounts Management

The more common actions are to add or remove a user account, or set the password for an account.

adduser username - creates a new account and home directory for the username. The account is effectively locked until the password is set.

rmuser username - removes the account but the user's home directory is retained. The rm command can be used to remove the directory if desired.

passwd [username] - sets the password of the current logged in user when no username is specified, or the named username when specified. Only root can change other user's passwords.

Address Map	Standard/Graphics I	Text 40	Text 80	Text 80x30		Bitmap
0	0	0	0	0		0
100	SIT	SIT	SIT	SIT		PDT
200	2FF					
300		3BF				
700			77F			
900				95F		
B00	0B00 to 0B1F CT					
F00	0F00-0F7F SAL	0F00-0F7F SAL	0F00-0F7F SAL	0F00-0F7F SAL		
1000	1000	1000	1000	1000		
1100	PDT	PDT	PDT	PDT		
1700	17FF	17FF	17FF	17FF		17FF
1800	1800	1800	1800	1800		1800
1900	May need a third set following this	May need a third set following this	May need a third set following this	May need a third set following this		GPU data and methods (bitmap mode)
1F00	1FFF	1FFF	1FFF	1FFF		1FFF
2000	2000	2000	2000	2000		2000
2100	GPU data and methods (all modes except bitmap)	GPU data and methods (all modes except bitmap)	GPU data and methods (all modes except bitmap)	GPU data and methods (all modes except bitmap)		CT
3300	33FF	33FF	33FF	33FF		
3400	3400	3400	3400	3400		
3500	File Data	File Data	File Data	File Data		
3700						37FF
3800						3800
3900						SIT
3A00						3AFF
3B00						3B00-3B7F SAL
3F00	3FF6	3FF6	3FF6	3FF6		

To create a new user account, perform the following steps:

- Login as root
- Execute the following commands in the shell:

```
$ adduser username          replace username with the desired name
$ passwd username           the system will prompt for the password to be entered twice
$ logout
```
- Login as the new user

4.6. Printer Configuration

Printing with TIPI and TI RS232 card devices is supported. The file /etc/printcap is used to configure named printers. When printing with the lp command and the -d printer option is not used, the printer selected will be “lp”. As such it’s necessary to have a printer named lp.

The format within /etc/printcap is standard; however, the format itself is not user friendly. Thus caution should be taken. The auto recommends first making a backup copy of the file before modifying.

The initialized values in the file are as follows and provide support for all parallel devices. As such it is not critical that the user modify /etc/printcap.

```
lp|lp1|local_printer:\
    :lp=/PIO/1:\
    :sd=/var/spool/lpd/lp:\
    :lf=/var/log/lpd-errs:\
    :mx#0:sh:sf
Lp|Lp1|Local_printer:lp=/PIO/1:sd=/var/spool/lpd/lp:lf=/var/log/lpd-errs:mx#0:sh
lp2|local_printer2:lp=/PIO/2:sd=/var/spool/lpd/lp2:lf=/var/log/lpd-errs:mx#0:sh
lp3|local_printer3:lp=/PI/PIO:sd=/var/spool/lpd/lp3:lf=/var/log/lpd-errs:mx#0:sh
```

Only the printer name field (the first) and the lp= option are used at this time, although the other valid fields can be present even though ignored. The printer name field support aliases by using the bar symbol. Fields are separated by colons. Fields can be written to separate lines via a colon and backslash character.

To select the default printer, lp, modify the file such that your desired printer is named lp.
All other names are at the user’s discretion.

For example, if the only printer is TIPI, you may delete the first three entries and modify the last to

```
lp|local_printer:lp=/PI/PIO:sd=/var/spool/lpd/lp3:lf=/var/log/lpd-errs:mx#0:sh
```

5. Architecture

5.1. Runtime

The runtime architecture has a boot loader, kernel, library methods, and processes.

The boot loader is allocated to cartridge ROM and responsible for initial system bringup.

The kernel is also allocated to cartridge ROM and responsible for managing common functions such as file management, process management.

Process management handles program execution, maintaining the queue of programs that must be executed, and starting the next required program after one terminates. When a login session is not active and no other programs are in the queue for execution, the login program is executed, running as root, to allow a user to login using their credentials. When a login session is active, and programs are queued, the next will be started. When no programs are queued to execute, the user's command shell is executed. This relatively simple logic works for most situations but has the disadvantage that "calling" programs, such as the shell, are restarted after the called program terminates, and any needed state information must be intentionally persisted between execution sessions. Release 4 is intended to correct this by supporting multi-processing.

The default shell, sh, provides a command interface that supports program chaining, pipes, redirection, and command aliases. This shell should by no means be confused with the standard shells sh, bash, etc. It may be better named "fbsh" for fairy-basic-shell. It has just enough. The cool part, though, is that anyone can write a shell and it can be easily included in the Unix99 distribution. Shell paths are set in /etc/passwd on a per user basis. Thus it is possible to have multiple shells available in the same platform.

Basic Unix commands are covered in a later section.

The system state, which includes the display mode, fonts, and colors, is preserved between program execution. Files are are automatically closed when a program terminates, although it is good programming practice to close files directly.

5.2. Library Methods

Library methods can be called directly by name, as is the standard.

Due to bloating of the executables, methods that were already used in the cartridge ROM and any others that could be "stuffed in" are made available to programs by allowing the programs to simply prepend "dylib." before calling those methods. This works as dylib is a structure of pointers to methods in cartridge ROM. There's one level of indirection which costs a small number of CPU cycles for each call.

As cartridge ROM paging has been integrated, more methods have been added to the cartridge ROMs. The dylib approach is in process of being deprecated in favor of program RAM-based trampoline methods that call to the appropriate ROM resident methods. The trampoline methods bear the name of the desired function but they internally set the ROM page, call the ROM-based method, handle any returns, and reset the ROM page to the default page.

The migration to ROM-based library methods is incomplete. As such program must bear direct linking within their image when the desired method isn't located in ROM. The largest methods, however, are located in ROM and available via one of the above three approaches.

In general there is a cost to any of the above approaches and it comes down to the need. The fastest code will always be the direct calls. There's little value to placing certain methods in ROM, such as 'isdigit()' as the processing cost, bloat of the added trampoline method, and perhaps more makes this prohibitive. The most bang for the buck comes by moving methods into ROM that already consume a lot of time and space, thus the added overhead is minimal.

5.3. Memory Organization and Usage

The following sections describe the Unix99 memory organization and usage.

5.3.1. RAM

Upper Memory

0xFFFF End of HEAP
…
Start of HEAP
End of TEXT segment
…
0xA000 TEXT segment

Lower Memory

0x3FFF Bottom-end of STACK
…
0x3400 Top-end of STACK
0x33FF End of lower memory HEAP
…
Start of lower memory HEAP
End of BSS
…
Start of BSS
End of DATA segment
…
0x2500 Start of DATA segment
0x2000 Reserved for Unix99 state and process data

Generally programs have available up to 24 KB RAM for program (text segment) in upper memory, up to 3 KB for the data and BSS segments, and a 4 KB stack. The heap consists of all remaining available memory, both lower and upper memory.

5.3.2. SRAM Organization

8300-831F	Registers
8320-833B	Reserved for Unix99 state and process data
833C-8357	Reserved for Unix99 speech data

Other addresses required for console ROM usage are used by Unix99, such as floating point support, DSR level 2 subprograms, etc.

5.3.3. VDP RAM Organization

The following describes the VDP RAM organization within the Unix99 environment. The Standard, Text 40, Text 80 and Text 80x30 provide the most flexibility, in that they full support text and file functions while the Bitmap mode does not. Programs can use bitmap mode and then change to one of the other modes to reenable text and file processing.

5.4. Persistent File Types

Persistent files store text, binary data and program images. A compromise approach for file handling per Unix style has been implemented, one that provides regular Unix file (stream) type access but also support for the long existing TI ecosystem. Programs are stored in binary formatted files.

Text file support is provided by a small set of methods, including open, fgets, fputs, fprintf, close and more. Full stream support is provided, but TI file types mapped as DISPLAY, VARIABLE, SEQUENTIAL can be used. For existing TI text files, the record size is obtained from the file specification, allowing for maximum compatibility. When creating files, Unix99 will use record size 80.

Binary file access is provided by a similar set of methods, including open, fread, fwrite, fseek, ftell, rewind and close, providing true stream access. They are mapped to INTERNAL, FIXED, 255, RELATIVE. In essence the fixed records are used as blocks and the methods internally map the stream to the blocks. The physical first block of the file is used to store file metadata, including the actual length of the stored data.

The binary files are essentially unix file streams. Any file method can be used with them, including any that are text oriented. For example, One could write a file with fwrite for 80 bytes, fputs a string, and fwrite again. Then read with fread for 80 bytes, fgets a string and fread the final.

fopen file modes “r”, “w”, and “a” pertain to TI text files.

fopen file modes “rb”, “rb+” and “wb” pertain to file streams, suitable for text and binary data.

6. Commands

This section describes commands available within Unix99. Many are standard Unix commands, some are examples and some are games.

6.1. Unix commands

In the interest of not documenting using multiple places, please use the ‘man’ command to read the manual reference for commands.

The usage of the ‘man’ command is as follows:
man program_name
where program_name is the command to be referenced.

To list all commands on the system, enter this command:
ls /bin

This will show a listing of all files in the /bin directory. Select one, such as hexdump, and enter the command:
man hexdump

This will present the manual page for hexdump.

The user is encouraged to read the manual pages and note the supported options. Modern unix commands have grown quite large, supporting many options. The bloat within them is outside of what is currently possible within the current Unix99 design without resorting to highly complex programming techniques.

Each user tends to grow accustomed to using a set of commands and options. In general the commands and options implemented in this initial Unix99 release represent those the author has most commonly used and are likely to heavily overlap with others’ usage. Those with the ability are encouraged to add new features and commands to the collection.

6.2. Non-Standard Commands

Many non-standard Unix commands are included in the /bin directory and provide support for TI-99 specific capabilities, such as setting colors, loading alternative fonts, and setting display mode. Manual pages exist for these as well.

6.3. Example Programs

Many example programs are included that demonstrate feature usage. They also serve as part of the regression test suite. A few are somewhat useful utilities, such as screen savers and image viewers.

These are included in the /example and /game directories.

7. Development Environment

The development environment consists of several tools. The compiler and linker are contained in a pre-built docker container named “cmcureau/tms9900-gcc”, built several years ago as a means to simplify the complex build and update instructions for GCC. This container is a Linux x86 image but does run on both x86 and ARM platforms. When used on ARM platforms it is significantly slower but still quite sufficient.

The source for Unix99 as well as several example programs are provided. Thus the user is free to modify any aspect and the source is provided as-is. Even so, please submit any useful modifications for inclusion in later releases.

Applications developed in this environment can be distributed without the operating system; however, it is recommended that a statement of version compatibility be provided.

Tools integrated into the development environment:
- GCC
- Libti99
- TI Image Tool (requires Java)
- xdt99 3.5 (requires Java)

The following instructions have been verified to work on MacOS Sequoia on x86 and ARM Apple silicon. They should also work in Linux.

7.1. Docker Container

Install Docker Desktop for your build machine. It can be downloaded at https://www.docker.com/products/docker-desktop.

Install the docker container named “cmcureau/tms9900-gcc”. This container provides the compiler, linker, make and other tools within the GCC collection.

7.2. First Application

A directory, “helloworld” is provided and serves as the template for any application the user desires to develop. A script “mc” (make/compile) calls make within the container, processes the products and moves the executable into the image directory and image files used for distribution.

To create a new application, simply copy the “helloworld" directory to a new directory with the desired name, update and build.

7.3. Standard C and Unix Libraries

All library elements described below generally conform to the MacOS, FreeBSD and Linux UNIX definitions, in that order. Descriptions for each can be found in the online manual pages on the internet or using man method, where method is the desired method name.

In the release code you will find the header directories, starting at the ‘include’ path. The methods described there are implemented and for use by application developers. These are the definitions of the application programmer interfaces (APIs). Note that ‘include’ also has a subordinate directory named ‘private’. All methods described there are internal, subject to change, etc. Their use is highly discouraged.

This release is by no means complete. It is currently one person’s work over the last two years with the intent to provide enough content to build truly usable and releasable software. If, as a developer, you come across an unimplemented method, write it yourself, but please submit it for inclusion so others can make use of it. If the need is more complex and requires a more complex feature set with architectural changes (example: shared memory), please feel free to make requests for support. In some cases the author will be willing to work with the requester.

A note concerning double precision literals and variables: The GCC compiler includes a defect where double precision literals are not correctly converted into the platform’s native format, radix-99. Libc methods for double precision values are included and working. atof() can be used to set literal values via string. A newer release of the compiler is on the way and will likely address this.

The standard APIs with at least partial implementation are:
- assert.h
- ctype.h
- dirent.h
- errno.h
- ioctl.h
- libgen.h
- signal.h
- statfs.h
- stdbool.h
- stddef.h
- stdio.h
- stdlib.h
- string.h
- time.h
- unistd.h
- utsname.h
- sys/ipc.h
- sys/shm.h
- sys/stat.h

7.4. TI-99/4a - Specific Libraries

The non-standard TI-99/4a specific library specifications are also in the ‘include’ directory. The headers contain more information than do the standard headers since there is currently no other documentation.

Again, if you as a developer decide an additional function is needed, please feel free to write it and send to the author for inclusion.

The APIs available are:
- cipher.h
- cache.h
- console.h
- constants.h
- conversion.h
- dylib.h
- gpu.h
- groff.h

- image.h
- math.h
- mem.h
- mouse.h
- rt.h
- sequencer.h
- soundqueue.h
- sounds.h
- speechqueue.h
- util.h
- version.h

8. System Directories

/[system disk]/bin
This directory contains the Unix99 executables.

/[system disk]/etc
This directory contains system configuration information.

/[system disk]/example
This directory contains example executables.

/[system disk]/fonts
This directory contains system fonts, including the default and alternatives.

/[system disk]/game
This directory contains games.

/[system disk]/home
This directory contains user home directories.

/[system disk]/proc
This directory is used for process-oriented files and should not be tampered with.

/[system disk]/tmp
This directory is used for temporary files.

/[system disk]/usr
This directory is used for user-oriented system directories and files. The manual pages are stored in /usr/share/man.

9. Future Release Plans

- Provide support for multi-threading and multi-processing by means of the expanded memory resources provided by the SAMS memory expansion.
- Complete set of libraries using trampolines

10. Credits

- GCC, targeted for the TI-99/4A by ‘insomnia’ - provides the C compiler for Unix99.
- libti99 by toursi - provides low-level file, sound, speech and VDP memory methods.
- fcmd by jedimatt42 - methodology for use of DSRs, use of RADIX 100 TI ROM methods, provisioning of string methods.

Without the above incredibly talented supporters in the AtariAge community, this work would not have been possible.

11. Troubleshooting

Files of the format “./*” are seen on the TI. Remove all files on your PC that are named “._*”. These files are part of the tar file and are included in the tar archive. Attempts will be made to remove these in future releases. The installation instructions have attempted to address this.

Some drives and devices are not accessible. Unix99 currently supports the commonly available devices and drives in the TI ecosystem. Other devices can be added but only tested devices have been made available. The program /example/dsr will list the DSRs on your system that are available.

Unix99 does not currently support aliasing / mapping of legacy DSRs for legacy software. The primary reason is that there is no legacy software running within Unix99.

Removing files doesn’t work on Classic99. Classic99 has a small delay where the deleted files can be listed after deletion. They will disappear after a few seconds.