ConvexOS V10.0
Advance Notice

Document No. 710-010030-002
October 1991

CONVEX Computer Corporation
Richardson, Texas USA

ConvexOS V10.0
Advance Notice

Document No. 710-010030-002

Printed in the United States of America

Contents

Appendix A tcsh Man Page

Overview

1

This Advance Notice is intended to make your transition to ConvexOS V10.0 as smooth as possible by providing you with information prior to the release. It describes new features and changes in the operating system and explains adjustments that must be made at your site.

Prerequisites

The V10.0 release of ConvexOS has the following prerequisites for C1 series, C2 series, and C3200 machines:

- If you are performing an upgrade, your system must be running V9.1 or V9.0 of ConvexOS.
- You must be running version 5.2 of SPU OS.
- If you have a C2 series CONVEX machine, you must install V3.7 of the diagnostic database. No upgrade of the diagnostic database is required for C1 series machines.
- You must be running the correct version of system diagnostics:
  - V6.6 of the C1 series system diagnostics
  - V3.5 of the C2 series system diagnostics
- The ConvexOS and Utilities V10.0 Installation Procedures will instruct you to create the following user and groups:
  - user lpr with UID 23
  - group lpr with GID 23
  - group batch with GID 24
  - group tape with GID 29

  Before you begin the installation, you should make sure that these UIDs and GIDs are not assigned to other users or groups.

Note

The installation procedure for ConvexOS V10.0 differs significantly from previous releases. Please read the *ConvexOS and Utilities V10.0 Installation Procedures* before you begin the installation process.

The ConvexOS V10.0 installation procedure requires a machine-specific activation key. The activation key will be shipped with the ConvexOS V10.0 installation tape on a single, separate sheet of paper. Be careful not to discard the activation key before you have completed the installation process.

Layered Products

The following table lists the versions of layered products that are compatible with ConvexOS and Utilities V10.0.

Product	Minimum V10.0-compatible version
CONVEX FORTRAN	V6.0
CONVEX C	V4.0
CONVEX Ada	V2.0
CONVEX Internet Services	V10.0
CONVEX NFS	V10.0
CONVEX CXbatch	V2.0
COVUEshell	V8.2
COVUEnet	V2.2
COVUEbatch	V2.1
COVUEedt	V1.1
COVUElib	V1.0
COVUEbinary	V1.0
CONVEX CXwindows	V2.1
CONVEX Consultant	V8.0, V8.1, V8.2, V9.0

All CONVEX Consultant utilities in V8.0, V8.1, and V8.2 will work successfully with ConvexOS V10.0 except for gprof. If gprof is in use at your site, you must install V9.0 of CONVEX Consultant.

Due to a change in the installation process, there are different minimum required versions for COVUE products installed *after* ConvexOS V10.0. If you install (or re-install) a COVUE product after you upgrade to ConvexOS V10.0, you must have the version listed in the following table. COVUE products already in place before the ConvexOS V10.0 upgrade will not be affected.

Product	Minimum version required if installed after ConvexOS V10.0
COVUEshell	V8.2.3
COVUEbatch	V2.1.2
COVUEedt	V1.2.1
COVUElib	V2.0.1
COVUEbinary	V1.0.1

Please note that these new versions represent only a change in the installation procedure, there are no functionality changes.

COVUEnet is not affected by this change.

Associated Documentation

The following documents are new with this release:

- *CONVEX Tape System User's Guide,* Second Edition
- *ConvexOS Tape System Quick Reference,* First Edition
- *ConvexOS* `dump` *and* `restore` *Quick Reference,* First Edition
- *CONVEX SPU System Manager's Guide,* First Edition
- *The ConvexOS Primer,* First Edition. This book replaces the *CONVEX UNIX Primer.*
- *The ConvexOS Extensions User Guide.* This book replaces
  - *CONVEX Checkpoint Restart Guide*
  - *CONVEX Share Concepts*
  - *CONVEX POSIX Concepts*

The following documents have been revised for this release:

- *ConvexOS Man Pages for Users,* Second Edition
- *ConvexOS Man Pages for Programmers,* Second Edition
- *ConvexOS Man Pages for System Managers,* Second Edition
- *Managing ConvexOS: Configuration Guide,* Second Edition
- *Managing ConvexOS: Operations Guide,* Second Edition
- *CONVEX Guide to Writing Device Drivers*
- *CONVEX POSIX Conformance,* Second Edition
- *CONVEX Networking Concepts,* Second Edition
- *ConvexOS Tutorial Papers,* Eighth Edition

The following documents have been previously published and are current for this release:

- *CONVEX Portable C User's Guide,* Fourth Edition
- *The C Programming Language,* First Edition. By Brian Kernighan and Dennis Ritchie, published by Prentice-Hall,
- *Programming perl,* First Edition. By Larry Wall and Randal L. Schwartz, published by O'Reilly and Associates, Inc.
- *GNU Emacs Manual,* Sixth Edition. By Richard Stallman, published by the Free Software Foundation
- *CONVEX Architecture Reference,* Fifth Edition
- *CONVEX adb Debugger User's Guide,* Sixth Edition
- *CONVEX Share,* First Edition
- *vi Quick Reference,* First Edition

The *CONVEX Assembly Language User's Guide* and the *CONVEX Loader User's Guide* are no longer bundled with releases of ConvexOS. They are now part of a new product, CONVEX ALL (Assembler, Loader and Libraries), and are contained in the *CONVEX Compiler Utilities User's Guides.* CONVEX ALL V1.0 will be distributed to all customers free of charge before the release of ConvexOS V10.0.

New Features

2

This chapter describes new features in ConvexOS V10.0.

Virtual Volume Manager (VVM)

The CONVEX Virtual Volume Manager improves the reliability of striped file systems through the use of data redundancy. There are two methods of redundancy: mirroring and parity.

Mirrored file systems maintain two copies of each stripe partition on different disks. If the primary disk fails, data can be retrieved from the second disk.

In parity file systems, parity information calculated from the other disks in the stripe is stored on each disk. If one of the disks fails, the data on that disk can be reconstructed using parity information.

VVM also supports "hot spares". You may designate certain disks or disk partitions to be hot spares. If hot spare disk space is available when a disk fails, VVM will automatically reconstruct the data from the failed disk onto the hot spare, and the hot spare will replace the failed disk in the stripe. If you do not have a hot spare, console error messages direct you to reconstruct the data manually.

For additional information, see Chapter 4, "Setting Up the Disk System" in *Managing ConvexOS: Configuration Guide.*

/etc/stripecap

The format of /etc/stripecap has changed with this release. Your existing /etc/stripecap file will be converted to the new format during the ConvexOS V10.0 installation process. (A copy will be kept in /etc/stripecap.old.)

For additional information about the new format, see the stripecap(5) man page.

vvmdaemon

`vvmdaemon` is the daemon for the Virtual Volume Manager. It must be started from /etc/rc.std.

`vvmdaemon` is responsible for

- reconstructing data once a disk failure has occurred
- restarting reconstruction operations that did not complete normally due to a system crash.

For additional information, see the vvmdaemon(8) man page.

New Utilities

Three new utilities are provided to help manage redundant partitions.

mvst

The `mvst` utility allows data from an existing stripe partition to be moved to a new partition. It may be used with redundant and non-redundant stripes.

You must be the superuser to use this utility.

For additional information, see the mvst(8) man page.

rmst

`rmst` deletes a stripe entry from kernel memory. It is also used to remove partitions from the hot spare list.

You must be the superuser to use this utility.

For additional information, see the rmst(8) man page.

qst

`qst` reports stripe information from /etc/stripecap for a specified disk device. For example:

```
# qst /dev/du2
/dev/du2h is used in /dev/st2 (redundant)
/dev/du2g is used in /dev/st4 (non-redundant)
```

You must be the superuser to use this utility.

Enhancements to Existing Stripe Utilities

Three stripe utilities have been enhanced to create and manage redundant partitions.

newst

The `-R` option to `newst` enables stripe redundancy. By default, a two-partition stripe will be mirrored; stripes with more than two partitions will use parity.

The `-H` option is used to add a specified partition to the hot spare list.

getst

The output of `getst` will include additional information for redundant stripes, as shown in this example:

```
# getst st0
stripe st0: redundant, sector size 2048 bytes, mounted on /usr/local
 section a: size 49200 Kbytes/partition, blocking factor 8 Kbytes
    partition 0: du6f (64, 1542) offset 0 Kbytes
    partition 1: du0f (64, 6) offset 0 Kbytes
    partition 2: du4a (64, 1025) offset 0 Kbytes
 section b: size 48600 Kbytes/partition, blocking factor 8 Kbytes
    partition 0: du6f (64, 1542) offset 49200 Kbytes
```

The `-H` option to `getst` may be used to display information about disk partitions in the hot spare list.

putst

The -p option to `putst` may be used to manually check the parity information on a redundant stripe.

Large Files Support

ConvexOS now supports files and file systems up to one terabyte in size. Files and file systems larger than 2 gigabytes are considered "large".

System Administration

Large file systems are created and mounted normally with `newfs`, `newst`, and `mount`.

By default, file systems are mounted with large file capability. This means that files greater than 2Gb may be created in them.

You can disable large file capability on a specific file system with the `nolf` option to `mount`. For example, the following line in /etc/fstab will cause /usr to be mounted without large file capability:

```
/dev/st2        /usr        4.2     rw,nolf    0 0
```

This option only prevents the creation of new large files. Large files that already exist in a file system may still be accessed in their entirety if the file system is later mounted with the `nolf` option.

Large file systems (file systems greater than 2Gb) are not required to be large file aware. Likewise, large files (files greater than 2Gb) that contain "holes" may reside in file systems that are not large. (For an explanation of holes, see the section titled "cp and mv" on page 10.)

Utility Support

Some utilities have been enhanced to support large files:

- `ls, cp, mv, tail, find`
- `ftp, rcp`
- `tar, cpio, pax` (for files up to 8Gb only)
- `dd, compact`
- `dump, xdump, restore`
- `chkpnt, restart`

Manipulating large files with other utilities may have serious consequences. Please refer to the "Restrictions" section below.

Programming Interface

Several system calls have been added or changed to handle large files. For additional information on the large files programming interface, refer to the *ConvexOS Extensions User Guide.*

Restrictions

Interactive editors such as `vi` and `Emacs` will truncate large files to 2Gb.

Shell redirection (<, >, >>) will not work properly on large files.

You may use other utilities with large files if you use pipes. For example, the command:

```
cat largefile | grep foo
```

will work on a large file while

```
grep foo largefile
```

will not.

The current NFS protocol does not support large files. As a result, only the first 2Gb of a large file can be accessed over NFS.

You will be unable to copy or move entire large files to a filesystem that has not been mounted with the `nolf` option. Both `cp` and `mv` will truncate the file to 2Gb.

The `-z` option has been added to `cp` and `mv` to preserve holes. When moving or copying large files, you may want to use this option. Please see the section titled "cp and mv" on page 10 for additional information.

Changes to Utilities and Procedures

3

This chapter describes changes to existing procedures and utilities.

Utilities

Changes have been made to several utilities.

GNU Emacs

Version 18.57 of GNU Emacs is now supported. This version mainly provides bug fixes. There is one significant functional change. The line-move function is now called move-line-internal. Initialization files that call line-move by name must be changed to call move-line-internal instead.

Perl

Version 4.03 of Perl is now supported. This is the version that is described in *Programming perl*, by Larry Wall and Randal L. Schwartz, which is distributed with ConvexOS. This version contains many bug fixes and some enhancements over the previous version.

`acctconv`

The `acctconv` utility, which was designed to convert accounting files to a new format for ConvexOS V8.1, is no longer supported.

`csh`

An enhanced version of `csh` is now supported. `tcsh`, which was originally developed at Ohio State University and is in the public domain, will be installed as /bin/csh. The previous version of `csh` will be available as /bin/oldcsh.

This version contains all the functionality of previous versions of `csh`, and has many new features, including

- File name completion
- `vi` and Emacs-like command line editing
- Terminal mode checking and resetting
- Spelling correction of command, file, and user names

For additional information on these new features, please refer to the `tcsh` man pages in Appendix A of this document. Please note, however, that `tcsh` will be installed as /bin/csh.

chkpnt

The checkpoint file format has been changed to support large files. As a result, checkpoint files created under ConvexOS V9.0 or V9.1 cannot be restarted after upgrading to ConvexOS V10.0.

cp and mv

The `-z` option has been added to both `cp` and `mv`. This option preserves "holes" in a file when it is copied or moved. A "hole" is region of a file that has not been written to but has been bypassed with the seek system call. The presence of the hole is recorded, but its content is not stored on disk and it does not occupy any disk space. A large file containing a hole may fit on a particular file system, while a file of the same size without the hole may not.

Without the `-z` option, both `cp` and `mv` will fill holes with zeros. A copy of a file may take up considerably more space than the original file. Likewise, a file may take up more space after it has been moved via `mv`.

For example, the file myfile has a size of 10 megabytes as reported by `ls`, but is only taking up 32 kilobytes of actual disk space, according to `du`:

```
% ls -l myfile
-rw------- 1 joe 10485760 Sep 10 15:26 myfile
% du -a myfile
32      myfile
```

This discrepancy occurs because myfile contains a hole. If myfile is copied or moved to another directory, the holes will be filled with zeros:

```
% cp myfile bigmyfile
% ls -l bigmyfile
-rw------- 1 joe 10485760 Sep 10 15:30 bigmyfile
% du -a bigmyfile
10304   bigmyfile
```

If myfile is copied with the `-z` option, the hole is preserved and the copy takes up only 32 kilobytes of actual disk space:

```
% cp -z myfile bigmyfile
% ls -l bigmyfile
-rw------- 1 joe 10485760 Sep 10 15:34 bigmyfile
% du -a bigmyfile
32      bigmyfile
```

chown and chgrp

`chown` now supports the `-R` flag, which causes it to recursively descend directory arguments, changing the owner as specified.

For example, the command

```
# /etc/chown -R joe .
```

will make user joe the owner of the current directory, and of all files and directories below the current directory.

chgrp also supports the -R option. The command

```
% chgrp -R staff .
```

will make staff the group for all files in the current directory, and for all files and directories below the current directory.

chown can also be used to change the owner and group of a file simultaneously. For example, the command

```
# chown joe.staff myfile
```

changes the owner of myfile to joe and the group to staff. The group and owner may also be a numerical GID or UID.

cron

There have been several enhancements to cron:

- World or group-writable .crontab or .cronrc files are now ignored by cron. Users with world or group-writable .crontab or .cronrc files will receive mail instructing them to execute one of the following commands:

  ```
  % chmod 644 .crontab
  ```

  ```
  % chmod 644 .cronrc
  ```

- The line

  ```
  CRONREPORT=1
  ```

  in a .cronrc file will indicate that the user would like to receive notification of output and failed requests via electronic mail. If cron is started at boot time with the -m option, CRONREPORT will be set for all users.

- cron now supports machine-specific .crontab files. Files named .crontab.*hostname* (where *hostname* is the name of the machine where the contents of the file should be executed) will be executed if they exist. If there is no crontab.*hostname* file, .crontab will be executed as usual. This feature can prevent multiple executions of .crontab files that are mounted on more than one machine via NFS.

make

The built-in make rules for generating executables from EFL (Extended FORTRAN Language) sources have been removed. make no longer has built-in rules for handling files ending in .e or .e,v.

Users who use make to perform operations on files with these suffixes may provide a rule for .e or .e,v files in their makefile.

man

man now supports a -g option, which will grep through all the man pages for a specified Perl regular expression. For example, the command

```
% man -g "crontab"
```

will report all the occurrences of the word "crontab" in the man page database.

This option will work with compressed man pages as well.

mkdir

The -p option has been added to `mkdir` to create a specified series of directories, rather than creating only one directory at a time. For example,

```
% mkdir -p dir1/dir2/dir3
```

creates directory dir1, directory dir1/dir2 and directory dir1/dir2/dir3, even though dir1 and dir2 did not exist when the command was invoked.

The -f option will prevent `mkdir` from reporting an error when one or more of the directories in the path already exist.

sh

/bin/sh has been upgraded to the AT&T System V, Release 2 version. The new version contains:

- shell functions
- restricted shell

as well as many bug fixes. This change should not affect existing `sh` scripts.

For additional information, see the sh(1) man page.

fsck

`fsck` has been upgraded to the 4.3BSD Tahoe release.

"Dirty bit" functionality has been added. When executed normally via `preen`, `fsck` only checks filesystems that were in use (marked "dirty") when the machine went down.

When executed from the command line, `fsck` checks a specified file system even if it is marked "clean".

The new -f option to `fsck` can be passed through `preen` to force checking on filesystems even if they are marked "clean":

```
# preen -f
```

This version also contains performance enhancements.

For additional information, see *Managing ConvexOS: Operations Guide* or the fsck(1) man page.

mount

The -F option has been added to mount to force mounting of a "dirty" file system. Normally, dirty bits will be cleared by `fsck`, so all the file systems encountered by mount will be clean.

Forcing the mount of a dirty file system with **`mount -F`** will also clear the dirty bit, but the file system will remain dirty. This action could eventually cause the system to panic, and is not recommended for use in normal circumstances.

lpr and lpd

Beginning with this release, the /usr/spool/lpd directory and its contents must be owned by user lpr and group lpr.

The ConvexOS and Utilities V10.0 Installation Procedures will instruct you to create user lpr with UID 23 and group lpr with GID 23. Before you begin the ConvexOS V10.0 installation, you should make sure that UID 23 and GID 23 are not in use.

The installation script will automatically change the owner and group for the /usr/spool/lpd directory and for all the printer queues within that directory.

opreq

Several changes have been made to `opreq`:

- Users no longer need to be the superuser to perform operator activities. Users that belong to the operator group (as specified in /etc/group) can execute the `select-done` and `select-cancel` commands. If you choose to take advantage of this feature, you must create an operator group and add the appropriate users to it.
- The interface has been improved. In addition to arrow keys, `vi` movement keys (h,j,k,l) and Emacs movement keys (CTRL-f, CTRL-b, CTRL-n, CTRL-p) may be used to move around the opreq screen.
- Mount, replace, and unmount messages can be logged in /usr/adm/opreq-acct. This file records the information about a message when `select-done` or `select-cancel` is performed. If you choose to log this information, you must create the file /usr/adm/opreq-acct. Like all accounting files, this file can get very large and should be trimmed periodically.
- Operators are notified when a tape request is satisfied via automatic volume recognition (a feature of the CONVEX ACS system). These messages are of type "Info" and will only be displayed if the .opreqrc file includes this message type.
- The status of "Info" and "Unmount" messages is changed to "Done" after two minutes. These messages are informative and don't require operator action.
- Multiple-tape mount requests now display all the tapes associated with a request simultaneously, so operators can retrieve all the necessary tapes at once.
- The `configure-drives` command has been added. This command allows the operator to enable or disable tape drives from within `opreq`.

For additional information about opreq, refer to Chapter 8, "Managing the Tape System", in *Managing ConvexOS: Operations Guide.*

crashdump

`crashdump` has been modified to work with 9-track, Rack Mount 3480, or DAT tape drives. Previously, `crashdump` only worked with 9-track drives.

If there is only one valid tape drive, `crashdump` will automatically use that drive. If there is more than one valid tape drive and one of those tape drives is a 9-track, that will be the default drive. If there is more than one valid tape drive, and none of them are 9-track, the user will be asked to select between valid drives. The `-S` option to `crashdump` will allow a user to specify a tape drive other than the default. When this option is specified, the user will be presented with a menu listing all available tape drives.

For additional information about `crashdump`, refer to Chapter 14 of *Managing ConvexOS: Operations Guide.*

crypt Library Routine

Due to export restrictions on DES encryption, the crypt library routine included in international distributions will not function correctly. If called, crypt will return an error.

This restriction does not apply to sites located within the United States or Canada.

CONVEX Share Scheduler

Shared groups are no longer supported by Share. In a shared group, the name of the group is identical to the name of one of the real users in the group.

If you have shared groups, you will have to replace them with not-shared groups when you install the CONVEX Share Scheduler V10.0. Utilities will be provided to assist you with the procedure. Please read the CONVEX Share Scheduler V10.0 Release Notice before you begin to install Share.

CONVEX Internet Services

In this release, the Maximum Transmission Unit (MTU) of the HYPERchannel network interface driver has been increased from 32 Kbytes to approximately 64 Kbytes. The larger MTU results in higher speed bulk data transfers over the HYPERchannel network interface. The MTU is set on a per-route basis using the `hyroute` command.

CONVEX NFS

There are two major enhancements to CONVEX NFS.

lockd

The lock manager system has been upgraded to the ONC 4.1 version. With the new lock manager, local file locking is done entirely within the kernel. The lock daemon (lockd) is no longer required to be running for local files locks to operate properly.

NFS Installation

Installs of NFS at domestic (United States and Canada) sites will automatically `sysgen` modules necessary for secure NFS functionality into vmunix and install the secure NFS utilities. Secure NFS is no longer a separate installation item.

Due to export restrictions on DES encryption, Secure NFS is not available in international distributions.

Tape System Enhancements

The ConvexOS Tape System now includes support for ANSI or IBM labeled tapes. For additional information, refer to the *ConvexOS Tape System User's Guide.*

Kernel Source Tree Modifications

ConvexOS V10.0 contains changes to the structure of the kernel source tree.

These modifications affect:

- /sys files
- /usr/include/sys files
- /usr/68k/include files
- `sysgen`

If you have user written device drivers that require these files, or if you have modified the kernel via sysgen, please read this section before installing ConvexOS V10.0. If you require assistance contact the Convex Technical Assistance Center (TAC).

Please note that sysgened layered and optional products must be reinstalled after installing ConvexOS V10.0.

Backward binary compatibility has been maintained for all programs.

Compatibility

This section describes ramifications on new kernel source and new kernel binary objects.

Kernel source compatibility

The CPU source portion of the ConvexOS V10.0 kernel is now compiled in the extended ANSI C mode (-ext) of the compiler. New CPU source that is added to the kernel must be ANSI C compliant.

The structure and organization of the kernel include files have changed with the ConvexOS V10.0 release, but the content of files that are needed by new kernel source has been maintained.

The include files that were previously in the following directories:

- /sys/dev_ccu
- /sys/dev_hsp
- /sys/dev_iop
- /sys/dev_viop
- /sys/kern20
- /sys/kern68k
- /sys/kernhsp
- /sys/kerniop
- /sys/kernviop
- /sys/mbs

have been moved to the directories:

- /sys/io/interfaces/ccu_if and its subdirectories
- /sys/io/interfaces/msg_if/mbs and its subdirectories
- /sys/io/lib

The contents of these files have been maintained as much as possible.

Return value from device drive close routine

The V10.0 kernel now respects the return value from a device driver close routine and passes it back to the user as the value of errno after the close() system call. User written device drivers must be sure to explicitly return a value (return zero if successful) and avoid "falling off the end" of the driver close function, which may be interpreted incorrectly as a non-zero return value.

Kernel binary object compatibility

A binary object that was compiled for use with ConvexOS V9.1 or earlier may not work with ConvexOS V10.0. You should only put objects that have been compiled specifically for or have been thoroughly tested with ConvexOS V10.0 into the ConvexOS V10.0 kernel.

---

sysgen

The order of `make` commands used to generate a custom ConvexOS has changed. Beginning with this release, `make` commands must be invoked in system-generated directories one at a time, and must be allowed to finish before additional `make` commands are started.

For additional information, see Chapter 16," Generating System Images" in *Managing ConvexOS: Configuration Guide.*

/sys Files

Due to the structure changes in the kernel source tree, the following directories in /sys have been obsoleted:

- /sys/cmi
- /sys/dev_ccu
- /sys/dev_hsp
- /sys/dev_iop
- /sys/kern20
- /sys/kern68k
- /sys/kernhsp
- /sys/kerniop
- /sys/kernviop
- /sys/mbs

These directories are replaced by

- /sys/io and its subdirectories
- /sys/kio
- /sys/netif

/usr/include Files

The structure and organization of the system include files have changed with the ConvexOS V10.0 release, but the user-visible content of files has been maintained as much as possible.

Some programs which include system header files may require changes to the include file set in order to recompile successfully. In general, this will only affect programs that are highly dependent upon the operating system and its internals.

As a result of the structure changes of the kernel source tree, the following symbolic links have been obsoleted:

- /usr/include/cmi
- /usr/include/dev_ccu
- /usr/include/dev_hsp
- /usr/include/dev_viop
- /usr/include/kern20
- /usr/include/kern68k
- /usr/include/kernhsp
- /usr/include/kerniop
- /usr/include/kernviop

Previously, /usr/include/dev_iop and /usr/include/mbs were symbolic links. They have been replaced by directories of the same name.

Also, symbolic links from

- /usr/include/kio to /sys/kio
- /usr/include/interfaces to /sys/io/interfaces

have been added.

/usr/68k/include Files

The structure and organization of the 68000 Tools include files have changed with the ConvexOS V10.0 release, but the user-visible content of files has been maintained as much as possible.Some programs which include 68000 Tools header files may require changes to the include file set in order to recompile successfully. In general, this will only affect programs that are highly dependent on the operating system and its internals.

The following symbolic links have been obsoleted:

- /usr/68k/include/dev_ccu
- /usr/68k/include/dev_hsp
- /usr/68k/include/dev_viop
- /usr/68k/include/kern20
- /usr/68k/include/kern68k
- /usr/68k/include/kernhsp
- /usr/68k/include/kerniop
- /usr/68k/include/kernviop

Previously, /usr/68k/include/dev_iop and /usr/68k/include/mbs were symbolic links. They have been replaced by directories of the same name.

tcsh Man Page

A

This appendix contains the man page for `tcsh`, an enhanced version of `csh` that was originally developed at Ohio State and is currently in the public domain. With the release of ConvexOS V10.0, this program will be installed as /bin/csh.

The man page included here describes only the enhancements made to `csh`. It will be useful to users who are already familiar with `csh`. The `csh` man page that will release with ConvexOS V10.0 will contain standard `csh` information as well as information concerning these enhancements.

Please note that this program will be installed as /bin/csh.

NAME

tcsh – C shell with file name completion and command line editing

SYNOPSIS

tcsh [**–bcdefilmnqstvVxX**] [*argument* . . .]

DESCRIPTION

Tcsh is an enhanced version of the Berkeley UNIX C shell *csh*(1). It behaves exactly like the C shell, except for the added utilities of:

1) Command line editing using Emacs-style commands.

2) Visual step up/down through the history list.

3) Terminal mode sanity checking and resetting.

4) Interactive command, file name and user name completion.

5) File/directory/user list in the middle of a typed command.

6) Spelling correction of command, file, and user names.

7) Lookup of command documentation in the middle of a typed command.

8) Enhanced history mechanism.

9) Automatic logout after long periods of idle time.

10) Automatic execution of a single command prior to printing each prompt.

11) Automatic periodic command execution.

12) A new syntax for the prompt, and the ability to set the prompt for "while" and "for" loops.

13) Time stamps in the history list.

14) An addition to the syntax of filenames to access entries in the directory stack, and the ability treat symbolic links in a sane way when changing directories.

15) The ability to watch for logins and logouts by user or terminal on the machine.

16) A scheduled event list, which specifies commands which are to be executed at given times.

17) A new builtin that does a subset of *ls*(1).

18) An addition to the file expression syntax for a character not in a set of characters and the ability to negate a globbing pattern.

19) New automatically initialized environment variables *HOST* and *HOSTTYPE*.

20) Commands for debugging terminal capabilities.

21) Searching for the visual history mechanism.

22) A new builtin for the *which*(1) command.

23) Restarting a stopped editor with two keystrokes.

24) Status line support

25) Automatic execution of a command when the current working directory is changed.

26) Native Language System support.

27) Automatic process time reporting.

28) OS Dependent Builtin Support

29) Automatic window size adjustment

30) Input files

31) Additional/Undocumented Options

32) Enhanced history/variable modifier expansion

For a description of standard C-shell features, see the *csh* manual page.

1. COMMAND LINE EDITING

Commands that the user types in may be edited using the same control characters that Gnu Emacs or vi uses. Arrow and function key sequences are also allowed. *Tcsh* allows this by setting the terminal to 'CBREAK' mode and reading the input one character at a time.

There is a new shell command, *bindkey,* that allows the user to redefine what any key does, or find out what any or all of the keys do.

Syntax: bindkey [-a] [-s] [-v] [-e] [-d] [-l] [-r] [--] [in-string [out-string | command]]

If no values are given all bindings are listed. If only in-string is given, bindings for the in-string is listed.

Otherwise it binds the in-string to the given out-string or command. If out-string, this is treated as input to *tcsh* when in-string is typed. This may be used recursively to currently a level of 10 deep.

There are two basic key maps: the normal and alternative one. The alternative is used by VI command mode. For multi-character input the basic key maps contains a sequence-lead-in for the first character in the input.

Options:

- -a bind in-string in alternative key map.
- -s bind an out-string instead of a command
- -v bind for default VI-mode
- -e bind for default emacs-mode
- -d bind to compiled in default
- -l list functions available with short description
- -r remove the binding of in-string

In strings control characters may be written as caret-<letter> and backslash ("\") is used to escape a character as follows:

- **\a** bell character
- **\n** line feed (new line)
- **\b** back space
- **\t** horizontal tab

\v	vertical tab
\f	form feed
\r	carriage return
\e	escape
\nnn	character code in octal

In all other cases \ escapes the following character. Needed for escaping the special meaning of \ and ^. Delete is written as "^?" (caret-question mark).

Tcsh always binds the arrow keys as defined by the termcap entry to:

up arrow	up-history
down arrow	down-history
right arrow	forward-char
left arrow	backward-char

except where these bindings would alter other single character bindings. If this is not desired one can avoid the automatic arrow key binding using *settc* to change the arrow key escape sequences to the empty string. The ansi/vt100 sequences for arrow keys are always bound.

The following is a list of the default emacs and vi bindings. Characters with the 8th bit set are written as M-<character>. Note however, that unlike with the old *bind* command (see below), the syntax M-<character> has no special meaning to the *bindkey* command, and the bindings for the sequence escape+<character> and M-<character> as given below are handled separately (although the the default bindings are the same). The printable ascii characters not mentioned in the list are bound to the *self-insert-command* function, which just inserts the given character literally into the input line. The remaining characters are bound to the *undefined-key* function, which only causes a beep (unless *nobeep* is set, of course).

EMACS bindings

"^@"	-> set-mark-command
"^A"	-> beginning-of-line
"^B"	-> backward-char
"^C"	-> tty-sigintr
"^D"	-> delete-char-or-list
"^E"	-> end-of-line
"^F"	-> forward-char
"^G"	-> is undefined
"^H"	-> backward-delete-char
"^I"	-> complete-word
"^J"	-> newline
"^K"	-> kill-line
"^L"	-> clear-screen
"^M"	-> newline
"^N"	-> down-history
"^O"	-> tty-flush-output
"^P"	-> up-history
"^Q"	-> tty-start-output
"^R"	-> redisplay
"^S"	-> tty-stop-output
"^T"	-> transpose-chars
"^U"	-> kill-whole-line
"^V"	-> quoted-insert
"^W"	-> kill-region
"^X"	-> sequence-lead-in

"^Y"	->	yank
"^Z"	->	tty-sigtsusp
"^["	->	sequence-lead-in
"^\"	->	tty-sigquit
"^]"	->	tty-dsusp
" " to "/"	->	self-insert-command
"0" to "9"	->	digit
":" to "~"	->	self-insert-command
"^?"	->	backward-delete-char

EMACS Multi-character and 8 bit bindings

"^[^D" or "M-^D"	->	list-choices
"^[^H" or "M-^H"	->	backward-delete-word
"^[^I" or "M-^I"	->	complete-word
"^[^L" or "M-^L"	->	clear-screen
"^[^Z" or "M-^Z"	->	run-fg-editor
"^[^[" or "M-^["	->	complete-word
"^[^_" or "M-^_"	->	copy-prev-word
"^[" or "M- "	->	expand-history
"^[!" or "M-!"	->	expand-history
"^[$" or "M-$"	->	spell-line
"^[0" or "M-0"	->	digit-argument
"^[1" or "M-1"	->	digit-argument
"^[2" or "M-2"	->	digit-argument
"^[3" or "M-3"	->	digit-argument
"^[4" or "M-4"	->	digit-argument
"^[5" or "M-5"	->	digit-argument
"^[6" or "M-6"	->	digit-argument
"^[7" or "M-7"	->	digit-argument
"^[8" or "M-8"	->	digit-argument
"^[9" or "M-9"	->	digit-argument
"^[?" or "M-?"	->	which-command
"^[B" or "M-B"	->	backward-word
"^[C" or "M-C"	->	capitalize-word
"^[D" or "M-D"	->	delete-word
"^[F" or "M-F"	->	forward-word
"^[H" or "M-H"	->	run-help
"^[L" or "M-L"	->	downcase-word
"^[N" or "M-N"	->	history-search-forward
"^[P" or "M-P"	->	history-search-backward
"^[R" or "M-R"	->	toggle-literal-history
"^[S" or "M-S"	->	spell-word
"^[U" or "M-U"	->	upcase-word
"^[W" or "M-W"	->	copy-region-as-kill
"^[_" or "M-_"	->	insert-last-word
"^[b" or "M-b"	->	backward-word
"^[c" or "M-c"	->	capitalize-word
"^[d" or "M-d"	->	delete-word
"^[f" or "M-f"	->	forward-word
"^[h" or "M-h"	->	run-help
"^[l" or "M-l"	->	downcase-word
"^[n" or "M-n"	->	history-search-forward
"^[p" or "M-p"	->	history-search-backward

"^[r" or "M-r"	-> toggle-literal-history
"^[s" or "M-s"	-> spell-word
"^[u" or "M-u"	-> upcase-word
"^[w" or "M-w"	-> copy-region-as-kill
"^[^?" or "M-^?"	-> backward-delete-word
"^X^X"	-> exchange-point-and-mark
"^X*"	-> expand-glob
"^X$"	-> expand-variables
"^XG"	-> list-glob
"^Xg"	-> list-glob

VI Insert Mode functions

"^C"	-> tty-sigintr
"^D"	-> list-or-eof
"^H"	-> backward-delete-char
"^I"	-> complete-word
"^J"	-> newline
"^K"	-> kill-line
"^L"	-> clear-screen
"^M"	-> newline
"^N"	-> is undefined
"^O"	-> tty-flush-output
"^P"	-> is undefined
"^Q"	-> tty-start-output
"^R"	-> redisplay
"^S"	-> tty-stop-output
"^T"	-> is undefined
"^U"	-> backward-kill-line
"^V"	-> quoted-insert
"^W"	-> backward-delete-word
"^X"	-> is undefined
"^Y"	-> tty-dsusp
"^Z"	-> tty-sigtsusp
"^["	-> vi-cmd-mode
"^\"	-> tty-sigquit
" " to "~"	-> self-insert-command
"^?"	-> backward-delete-char

VI Command Mode functions

"^@"	-> is undefined
"^A"	-> beginning-of-line
"^B"	-> is undefined
"^C"	-> tty-sigintr
"^D"	-> list-choices
"^E"	-> end-of-line
"^F"	-> is undefined
"^G"	-> list-glob
"^H"	-> backward-delete-char
"^I"	-> vi-cmd-mode-complete
"^J"	-> newline
"^K"	-> kill-line
"^L"	-> clear-screen

"^M"	->	newline
"^N"	->	down-history
"^O"	->	tty-flush-output
"^P"	->	up-history
"^Q"	->	tty-start-output
"^R"	->	redisplay
"^S"	->	tty-stop-output
"^T"	->	is undefined
"^U"	->	backward-kill-line
"^V"	->	is undefined
"^W"	->	backward-delete-word
"^X"	->	expand-line
"^["	->	sequence-lead-in
"^\"	->	tty-sigquit
" "	->	forward-char
"!"	->	expand-history
"$"	->	end-of-line
"*"	->	expand-glob
"0"	->	vi-zero
"1" to "9"	->	digit-argument
"?"	->	which-command
"@"	->	is undefined
"A"	->	vi-add-at-eol
"B"	->	backward-word
"C"	->	vi-chg-to-eol
"D"	->	kill-line
"I"	->	vi-insert-at-bol
"J"	->	history-search-forward
"K"	->	history-search-backward
"O"	->	sequence-lead-in
"R"	->	vi-replace-mode
"S"	->	vi-substitute-line
"T"	->	toggle-literal-history
"V"	->	expand-variables
"W"	->	forward-word
"X"	->	backward-delete-char
"["	->	sequence-lead-in
"\^"	->	beginning-of-line
"a"	->	vi-add
"b"	->	backward-word
"c"	->	is undefined
"d"	->	delete-word
"h"	->	backward-char
"i"	->	vi-insert
"j"	->	down-history
"k"	->	up-history
"l"	->	forward-char
"r"	->	vi-replace-char
"s"	->	vi-substitute-char
"t"	->	toggle-literal-history
"v"	->	expand-variables
"w"	->	vi-beginning-of-next-word
"x"	->	delete-char

```
"~"                     -> change-case
"^?"                    -> backward-delete-char
"M-?"                   -> run-help
"M-["                   -> sequence-lead-in
"M-O"                   -> sequence-lead-in
VI Multi-character bindings

"^[?"                   -> run-help
```

There is also an older version of bindkey called *bind,* that allows the user to redefine what any key does, or find out what any or all of the keys do. This is retained for compatibility reasons.

If given two arguments *bind* binds the function (first argument) to the given key (second argument). The key may be: the direct character or a caret-<letter> combination, which is converted to control-<letter>; M-<letter> for an escaped character; or F-<string> for a function key. For the last of these, the function key prefix must be bound to the function "sequence-lead-in" and the string specified to the *bind* command must not include this prefix.

If given one argument *bind* takes the argument as the name for a key and tells what that key does. As a special case, the user can say

```
        bind emacs
```

or

```
        bind vi
```

to bind all the keys for Emacs or vi mode respectively.

If given no arguments *bind* tells what all of the keys do. If you give bind the single argument of 'defaults', it resets each key to its default value (see the above list).

2. VISUAL HISTORY

The keys ^P and ^N are used to step up and down the history list. If the user has typed in the following:

```
> ls
foo     bar
> echo mumble
mumble
>
```

then enters *^P,* the shell will place "echo mumble" into the editing buffer, and will put the cursor at the end of the line. If another *^P* is entered, then the editing line will change to "ls". More *^P*s will make the bell ring, since there are no more lines in the history. *^N* works the same way, except it steps down (forward in time).

An easy way to re-do a command is to type *^P* followed by *Return.* Also, pieces of previous commands can be assembled to make a new command. The commands that work on regions are especially useful for this.

^P and *^N* actually only copy commands from out of the history into the edit buffer; thus the user may step back into the history and then edit things, but those changes do not affect what is actually in *tcsh 's* history.

Another way to recall (parts of) history commands is via the 'expand-history' function. A variation of the 'expand-history' function is called 'magic-space'. This function expands *csh* history, and always appends a space. Magic-space thus can be bound to <space>, to automatically expand *csh* history. Expand-history is normally bound to *M-<space>* and magic-space is not

bound.

3. TTY MODE SANITY

As part of the editor, *tcsh* does a check and reset of the terminal mode bits. If the speed has been changed, then *tcsh* will change to using that speed. *Tcsh* will also obey changes in the padding needed by the tty. Some changes to the command keys will be obeyed, however if a command key is unset, *tcsh* will reset it to what it was. Also, the shell will automatically turn off RAW and CBREAK (on systems that use *termio(7)* it will turn on ICANON) modes, and will turn on the tty driver's output processing (OPOST).

The list of the tty modes that are always set or cleared by *tcsh* can be examined and modified using the *setty* builtin. The *setty* display is similar to *stty(1)*, and varies depending on the system's tty driver. Modes that *tcsh* will always try to set are shown as *+mode* modes that *tcsh* will always try to clear are shown as *-mode* and modes that tcsh will track and allow to be modified are not shown by default, or if the *-a* flag is given, are shown without a leading sign. For example if one wants to set the *echok* flag and let the *echoe* pass unchanged:

```
> setty
iflag:-inlcr -igncr +icrnl
oflag:+opost +onlcr -onlret
cflag:
lflag:+isig +icanon +echo +echoe -echok -echonl -noflsh
    +echoctl -flusho +iexten
> setty +echok echoe
> setty
iflag:-inlcr -igncr +icrnl
oflag:+opost +onlcr -onlret
cflag:
lflag:+isig +icanon +echo +echok -echonl -noflsh +echoctl
    -flusho +iexten
```

4. WORD COMPLETION

In typing commands, it is no longer necessary to type a complete name, only a unique abbreviation is necessary. When you type a TAB to *tcsh* it will complete the name for you, echoing the full name on the terminal (and entering it into the edit buffer). If the prefix you typed matches no name, the terminal bell is rung, unless the variable *nobeep* is set. The name may be partially completed if the prefix matches several longer names. If this is the case, the name is extended up to the point of ambiguity, and the bell is rung. This works for file names, command names, shell variables and the ~ user name convention. The variable *fignore* may be set to a list of suffixes to be disregarded during completion.

Example

Assume the current directory contained the files:

```
DSC.TXT     bin     cmd     lib     memos
DSC.NEW     chaos   cmtest  mail    netnews
bench  class dev    mbox    new
```

The command:

```
> gnumacs ch[TAB]
```

would cause *tcsh* to complete the command with the file name chaos. If instead, the user had typed:

```
> gnumacs D[TAB]
```

tcsh would have extended the name to DSC and rung the terminal bell, indicating partial completion. However, if *fignore* had previously been set to a list containing .NEW as one element, e.g. (.o .NEW), *tcsh* would have completed the 'D' to DSC.TXT.

File name completion works equally well when other directories are addressed. Additionally, *tcsh* understands the C shell tilde (~) convention for home directories. Thus,

```
> cd ~speech/data/fr[TAB]
```

does what one might expect. This may also be used to expand login names only. Thus,

```
> cd ~sy[TAB]
```

expands to

```
> cd ~synthesis
```

Command names may also be completed, for example,

```
> gnum[TAB]
```

will expand to "gnumacs" (assuming that there are no other commands that begin with "gnum").

Shell and environment variables are recognized also and in addition they can be expanded:

```
> set local=/usr/local
> echo $lo[TAB]
```

will expand to "$local/". Note that a slash is appended because the expanded variable points to a directory. Also:

```
> set local=/usr/local
> echo $local/[^D]
bin/ etc/ lib/ man/ src/
```

will correctly list the contents of /usr/local. Shell and environment variables can also be expanded via the expand-variables function:

```
> echo $local/[^X$]
> echo /usr/local/
```

Completion also works when the cursor is in the middle of the line, rather than just the end. All of the text after the cursor will be saved, the completion will work (possibly adding to the current name), and then the saved text will be restored in place, after the cursor.

The behavior of the completion can be changed by the setting of several shell variables:

Setting the *recexact* variable makes an exact command be expanded rather than just ringing the bell. For example, assume the current directory has two subdirectories called foo and food, then with *recexact* set the following could be done:

```
> cd fo[TAB]
```

to ...

> cd foo[TAB]

to ...

> cd foo/

rather than beeping on the second TAB.

If the *autolist* variable is set, invoking completion when several choices are possible will automatically list the choices, effectively merging the functionality described in the next section into the completion mechanism. The "noise level" can be controlled by the value that *matchbeep* is set to: With matchbeep=nomatch, completion will only beep if there are no matching names; with matchbeep=ambiguous, completion will *also* beep if there are many possible matches; with matchbeep=notunique, completion will *also* beep when there is an exact match but there are other, longer, matches (see recexact). With matchbeep=never or set to any other value completion will never beep. If matchbeep is not set it defaults to ambiquous.

If the *autoexpand* variable is set, the expand-history function will be invoked automatically before the completion attempt, expanding normal *csh* history substitutions.

For covert operation, the variable *nobeep* can be set; it will prevent the completion mechanism, as well as *tcsh* in general, from actually beeping. Finally, if the *autocorrect* variable is set, the spelling correction is attempted for any path components up to the completion point.

5. LISTING OF POSSIBLE NAMES

At any point in typing a command, you may request "what names are available". Thus, when you have typed, perhaps:

> cd ~speech/data/fritz/

you may wish to know what files or subdirectories exist (in ~speech/data/fritz), without, of course, aborting the command you are typing. Typing the character Control-D (^D), will list the names (files, in this case) available. The files are listed in multicolumn format, sorted columnwise. Directories are indicated with a trailing '/', executable files with a '*', symbolic links with a '@', sockets with a '=', FIFOs (named pipes) with a '|', character devices with a '%', and block devices with a '#'. Once printed, the command is re-echoed for you to complete.

Additionally, one may want to know which files match a prefix. If the user had typed:

> cd ~speech/data/fr[^D]

all files and subdirectories whose prefix was "fr" would be printed. Notice that the example before was simply a degenerate case of this with a null trailing file name. (The null string is a prefix of all strings.) Notice also, that a trailing slash is required to pass to a new directory for both file name completion and listing.

The degenerate

> ~[^D]

will print a full list of login names on the current system. Note, however, that the degenerate

> <Spaces>[^D]

does not list all of the commands, but only beeps.

Listing/expanding of words that match a name containing wildcard characters can be done via the list-glob/expand-glob function:

> ls

```
foo.c  bar.c  a.out
> vi *.c[^Xg]
foo.c bar.c
> vi *.c[^X*]
> vi foo.c bar.c
```

Command Name Recognition

Command name recognition and completion works in the same manner as file name recognition and completion above. The current value of the environment variable *PATH* is used in searching for the command. For example

```
> newa[TAB]
```

might expand to

```
> newaliases
```

Also,

```
> new[^D]
```

would list all commands (along PATH) that begin with "new".

Note that Control-D has three different effects on *tcsh*. On an empty line (one that contains nothing, not even spaces), *^D* sends an EOF to *tcsh* just as it does for normal programs. When the cursor is in the middle of a line of text, *^D* deletes the character that the cursor is under. Finally, a *^D* at the end of a line of text lists the available names at that point. To get a list of available names when the cursor is in the middle of a line (or on an empty line), a Meta-Control-D should be typed (Escape followed by Control-D).

6. SPELLING CORRECTION

If while typing a command, the user mistypes or misspells a file name, user name, or command name, *tcsh* can correct the spelling. When correcting a file name, each part of the path is individually checked and corrected. Spelling correction can be invoked in several different ways:

The *spell-word* function, normally bound to M-s (and M-S), will attempt to correct the word immediately before the cursor. For example, suppose that the user has typed:

```
> cd /uxr/spol/news[ESC s]
```

Tcsh will check the path for spelling, correct the mistakes, and redraw the line as

```
> cd /usr/spool/news
```

leaving the cursor at the end of the line.

Spelling correction of the entire command line (independent of where the cursor is) can be done with the *spell-line* function, normally bound to M-$ (Escape Dollar-sign). It will check each word independently, but in order to avoid e.g. command options, no correction is attempted on words whose first character is found in the string "!.\^-*%".

Finally, automatic spelling correction will be done each time the Return key is hit, if the *correct* variable is set to an appropriate value: correct=cmd will cause the spelling of the command name only to be checked, while correct=all causes checking of all words on the line, like the spell-line function. If any part of the command line is corrected, the user will be given a special prompt as defined by the *prompt3* variable, followed by the corrected line, e.g.

```
> lz /usr/bin
CORRECT>ls /usr/bin (y/n)?
```

Answering 'y' or <space> at the prompt will cause the corrected line to be used, anything else will leave the original line unchanged.

Automatic correction is not guaranteed to work the way the user intended. Command line parsing is done in a rudimentary fashion. It is mostly provided as an experimental feature. Suggestions and improvements are welcome.

7. DOCUMENTATION LOOKUP

The editor function *run-help* (M-h) prints a help file on the current command (using the same definition of current as the completion routines use). This help file is found by searching the path list HPATH for files of the form foo.help, foo.1, foo.8, or foo.6 in that order (assuming that the current command is foo). The file is just printed, not paged in any way. This is because *run-help* is meant to be used to look up short help files, not manual pages (although it can do manual pages also).

8. ENHANCED HISTORY MECHANISM

Tcsh will save the history list between login sessions. It does this by writing the current list to the file "~/.history" on logout, and reading it in on login. For example, placing the line

```
> set history=25 savehist=20
```

tells *tcsh* to save the last 25 commands on the history list, and to save the last 20 of them between logins. The "savehist" variable may be set up to the size of history, although it is an error to have *savehist* larger than *history*. In addition to the above *tcsh*, keeps unparsed (literal) versions of the history if the variable *histlit* is set. Also the toggle-history function toggles between the parsed and literal version of the recalled history in the editor buffer. For example:

```
> set histlit
> echo !:s/foo/bar; ls
Modifier failed.
> ^P
> echo !:s/foo/bar; ls
> unset histlit
> echo !:s/foo/bar; ls
Modifier failed.
> ^P
> echo unset histlit[M-r]
> echo !:s/foo/bar; ls
```

Tcsh also supports the history escape *!#*. This undocumented *csh* escape holds the words of the current line. This is useful in renaming commands:

```
> mv foo bar!#:1
mv foo barfoo
```

Care should be taken when using this history expansion in *csh* since there is no check for recursion. In *tcsh* up to 10 levels of recursion are allowed.

Another difference between *tcsh* and *csh* history expansion, is the treatment of history arguments. In *csh !3d* expands to event 3 with the letter "d" appended to it. There is no way to repeat a command that begins with a number using the name of the command in the history escape. In *tcsh* only numeric arguments are treated as event numbers; therefore *!3d* is interpreted as: repeat the last command that started with the string "3d". To mimick the *csh* behavior *!3\d* can be used.

9. AUTOMATIC LOGOUT

The automatic logout time is controlled by the variable *autologout,* the value of which is the number of minutes of inactivity will be allowed before automatically logging the user out. When that many minutes have been reached, the shell prints "autologout" and dies (without executing ~/.logout). The default for *tcsh* is to set *autologout* for 60 minutes on login shells, and when the user is root. To disable autologout (for instance in a window system), unset the shell variable *autologout.*

10. EXECUTION OF A COMMAND PRIOR TO EACH PROMPT

Tcsh supports a special alias, *precmd,* which if set holds a command that will be executed before printing each prompt. For example, if the user has done

```
> alias precmd date
```

then the program *date* will be run just before the shell prompts for each command. There are no limitations on what precmd can be set to do, although discretion should be used.

11. PERIODIC COMMAND EXECUTION

Tcsh is now capable of providing periodic command execution through the use of the shell variable *tperiod* and the alias *periodic.* When these items are set, the alias *periodic* will be executed every *tperiod* minutes. This provides a convenient means for checking on common but infrequent changes, such as new messages. Example:

```
> set tperiod = 30
> alias periodic checknews
```

This will cause the *checknews*(1) program to be run every 30 minutes. Having the alias *periodic* set but with an unset *tperiod* (or a value of 0 for *tperiod)* will cause *periodic* to degenerate to another form of *precmd.*

12. NEW PROMPT FORMAT

The format for the *prompt* shell variable has been changed to include many new things, such as the current time of day, current working directory, etc.. The new format uses "%<char>" to signal an expansion, much like *printf(3S).* The available sequences are:

%d or %/	Current working directory.
%~	cwd. If it starts with $HOME, that part is replaced by a ~. In addition if a directory name prefix matches a user's home directory, that part of the directory will be substituted with ~user. NOTE: The ~user substitution will only happen if the shell has performed a ~ expansion for that user name in this session.
%c or %.	Trailing component of cwd, may be followed by by a digit to get more than one component, if it starts with $HOME, that part is replaced with a ~.
%C	Trailing component of cwd, may be followed by a digit to get more than one component, no ~ substitution.
%h, %!, !	Current history event number.
%M	The full machine hostname.
%m	The hostname up to the first ".".
%S (%s)	Start (stop) standout mode.
%B (%b)	Start (stop) boldfacing mode. (Only if

	tcsh was compiled to be eight bit clean.)
%U (%u)	Start (stop) underline mode. (Only if tcsh was compiled to be eight bit clean.)
%t or %@	Current time of day, in 12-hour, am/pm format.
%T	Current time of day, in 24-hour format. (But see the *ampm* shell variable below.)
\c	'c' is parsed the same way as in bindkey.
^c	'c' is parsed the same way as in bindkey.
%%	A single %.
%n	The user name, contents of $user.
%w	The date in <Mon> dd format.
%W	The date in mm/dd/yy format.
%D	The date in yy-mm-dd format.
%l	The line (tty) the user is logged on.
%L	clear from prompt to end of display or end of line.
%#	A '#' if tcsh is run as a root shell, a '>' if not.
%{..%}	Include string as a literal escape sequence. Note that the enclosed escape sequence, should only be used to change terminal attributes and should not move the cursor location. Also, this cannot be the last character in the prompt string. (Available only if tcsh was compiled to be eight bit clean.)
%?	return code of the last command executed just before the prompt.
%R	In prompt3 this is the corrected string; in prompt2 it is the status of the parser.

The sequences for standout are often used to indicate that this is an enabled (running as root) shell. An example:

```
> set prompt="%m [%h] %B[%@]%b [%/] you rang? "
tut [37] [2:54pm] [/usr/accts/sys] you rang? _
```

In addition, there is a new variable, *prompt2,* which is used to prompt for the body of while and for loops (wherever normal *csh* prompts with a question mark). The default for *prompt2* is "%R? ": the status of the parser followed by a question mark. This alternate prompt is also used when the parser is waiting for more input; i.e. when the previous line ended in a \. The *prompt3* variable is used when displaying the corrected command line when automatic spelling correction is in effect; it defaults to "CORRECT>%R (y|n)? ".

13. TIME-STAMPED HISTORY LIST

The history list in *tcsh* now has a time-of-day stamp attached to each history list event. This time stamp is printed whenever the history command is executed. This allows the user to keep track of when the various events occurred. The time stamps are not maintained on the saved history list (also available in *csh*); thus, on logging back in, all the saved history events will be recorded with the login time as their time stamp. The time stamp printouts can be omitted from the history list by adding the -t switch to the *history* command.

14. DIRECTORY ACCESS

Tcsh supports three new flags to control directory style printing for cd, pushd, popd, and dirs:

-n Print entries in new lines so that the screen width is not exceeded

-l Don't print ~ but print the whole path

-v Print the stack entries one in each line, preceeded by the stack number.

Note that popd +n can be used to pop out stack entries of directories that do not exist any more.

Tcsh will now allow the user to access all elements in the directory stack directly. The syntax "=<digit>" is recognized by *tcsh* as indicating a particular directory in the stack. (This works for the file/command name recognition as well.) This syntax is analogous to the ~ syntax for access to users' home directories. The stack is viewed as zero-based, i.e., =0 is the same as $cwd, which is the same as ".". As a special case, the string "=-" is recognized as indicating the last directory in the stack. Thus,

```
> dirs -v
0 /usr/net/bin
1 /usr/spool/uucp
2 /usr/accts/sys
> echo =2
/usr/accts/sys
> ls -l =1/LOGFILE
-rw-r--r-- 1 uucp        2594 Jan 19 09:09 /usr/spool/uucp/LOGFILE
> echo =-/.cs*
/usr/accts/sys/.cshrc
> echo =4
Not that many dir stack entries.
>
```

Tcsh will complain if you ask for a directory stack item which does not exist.

In the normal *csh*, saying "pushd +2" would rotate the entire stack around through 2 stack elements, placing the entry found there at the top of the stack. If, however, the new shell variable *dextract* is set, then issuing "pushd +n" will cause the nth directory stack element to be extracted from its current position, which will then be pushed onto the top of the stack. Example:

```
> dirs
~ /usr/spool/uucp /usr/net/bin /sys/src
> set dextract
> pushd +2
/usr/net/bin ~ /usr/spool/uucp /sys/src
> unset dextract
> pushd +2
/usr/spool/uucp /sys/src /usr/net/bin ~
```

The way symbolic links that point to directories are crossed is determined by two variables: *chase_symlinks* and *ignore_symlinks*. If *chase_symlinks* is set, then every time the directory changes, $*cwd* reflects the real directory name, and not the name through the link. A notable exception is the user's home directory, but that should be fixed. If *ignore_symlinks* is set, then directory change tries to find where you came from before you crossed the link to change the directory relatively. If you chdir through a symbolic link and then cd .., you will end .. relatively to where you were before you crossed the link and not .. relatively to where the symbolic link points.

For example:

```
> cd /tmp
> mkdir from from/src to
> ln -s ../from/src to/dst
```

```
> echo $cwd
/tmp

> unset ignore_symlinks; unset chase_symlinks
> cd to/dst; echo $cwd
/tmp/to/dst
> cd ..
/tmp/from

> unset ignore_symlinks; set chase_symlinks
> cd /tmp/to/dst; echo $cwd
/tmp/from/src
> cd ..; echo $cwd
/tmp/from

> set ignore_symlinks; unset chase_symlinks
> cd /tmp/to/dst; echo $cwd
/tmp/to/dst
> cd ..; echo $pwd
/tmp/to
```

In case you are wondering what happens when you set both, *ignore_symlinks* will override *chase_symlinks.*

15. WATCHING FOR LOGINS AND LOGOUTS

Tcsh has a mechanism so that the user can watch for login and logout activity of any user or terminal in the system. This is accomplished using the new special shell variable *watch,* which contains login/terminal name pairs to be checked for activity. For example:

```
> set watch=(sys ttyjd root console)
```

This setting will allow the user to check on when the user "sys" logs in on /dev/ttyjd. Similarly, it will inform the user of root's activity on the console. In order to be more general, the word "any" may be substituted for either a user's or a terminal's name, thus allowing

```
> set watch=(brad any any ttyh0)
```

which will check for user "brad" logging in or out of the system on any terminal, as well as anyone logging in to /dev/ttyh0. Naturally, the completely general case

```
> set watch=(any any)
```

allows the user to check on any and all login/logout activity in the the system.

By default, the interval between checks of users on the system is 10 minutes; this can be changed by making the first element of *watch* a number of minutes which should be used instead, as in

```
> set watch=(40 any any)
```

which will check for any users logging in or out every 40 minutes.

There is also a new command, *log,* which is used to cause *tcsh* to inform the user of all users/terminals affected by *watch* whether they have been announced before or not. This is useful if a user has been on for some time and cannot remember if a particular person/terminal is online right now or not. *Log* will reset all indication of previous announcement and give the user the

login list all over again, as well as printing the current value of *watch*.

The first time that *watch* is set at *tcsh* startup, all affected users and terminals will be printed as though those users/terminals had just logged on. This may appear to be a bug, but is generally considered a feature, since it allows the user to see who is on when he first logs in.

The format of the printouts can be tailored via setting of the variable *who*. The following sequences are available for the format specification:

%n	The name of the user that logged in/out.
%a	The observed action, i.e. "logged on", "logged off", or "replaced <olduser> on".
%l	The line (tty) the user is logged on.
%S (%s)	Start (stop) standout mode.
%B (%b)	Start (stop) boldfacing mode. (Only if tcsh was compiled to be eight bit clean)
%U (%u)	Start (stop) underline mode. (Only if tcsh was compiled to be eight bit clean)
%M	The full hostname of the remote host ("local" if non-remote).
%m	The hostname up to the first ".". If only the ip address is available or the utmp field contains the name of an x-windows display, the whole name is printed.
%t or %@	The time, in 12-hour, am/pm format (logout time is approximated if unavailable).
%T	The time, in 24-hour format. (but see the "ampm" shell variable below).
%w	The date in <Mon> dd format.
%W	The date in mm/dd/yy format.
%D	The date in yy-mm-dd format.

The %M and %m sequences are only available on systems that store the remote hostname in /etc/utmp. If *who* is not set, the format defaults to "%n has %a %l from %m.", or "%n has %a %l." on systems that don't store the hostname.

16. TIMED EVENT LIST

Tcsh now supports a scheduled-event list through the use of the command *sched*. This command gives the user a mechanism by which to arrange for other commands to be executed at given times. An event is added to the scheduled-event list by saying

```
> sched [+]hh:mm <command>
```

as in

```
> sched 11:00 echo It\'s eleven o\'clock.
```

This will make an entry in the list at 11am for the echo command to be run with the given arguments. The time may be specified in either absolute or relative time, and absolute times may have a morning/afternoon specification as well, using "am" or "pm." For example,

```
> sched +2:15 /usr/lib/uucp/uucico -r1 -sother
> sched 5pm set prompt='[%h] It\'s after 5; go home: >'
> sched +3am echo This syntax doesn\'t work.
Relative time inconsistent with am/pm.
```

```
>
```

Note that *tcsh* will complain if you try to make faulty time specifications.

Printing the current time-event list is accomplished by giving the *sched* command with no arguments:

```
> sched
    1  Wed Apr  4 15:42  /usr/lib/uucp/uucico -r1 -sother
    2  Wed Apr  4 17:00  set prompt=[%h] It's after 5; go home: >
>
```

There is also a mechanism by which the user can remove an item from the list:

```
> sched --3
Usage for delete: sched -<item#>.
> sched -3
Not that many scheduled events.
> sched -2
> sched
    1  Wed Apr  4 15:42  /usr/lib/uucp/uucico -r1 -sother
>
```

All commands specified on the scheduled-event list will be executed just prior to printing the first prompt immediately following the time when the command is to be run. Hence, it is possible to miss the exact time when the command is to be run, but *tcsh* will definitely get around to all commands which are overdue at its next prompt. Scheduled-event list items which come due while *tcsh* is waiting for user input will be executed immediately. In no case, however, will normal operation of already-running commands be interrupted so that a scheduled-event list element may be run.

This mechanism is similar to, but not the same as, the *at*(1) command on some Unix systems. Its major disadvantage is that it does not necessarily run a command at exactly the specified time (but only if another command is already being run). Its major advantage is that commands which run directly from *tcsh,* as sched commands are, have access to shell variables and other structures. This provides a mechanism for changing one's working environment based on the time of day.

17. BUILTIN FOR ls -F

There is a new builtin command called *ls-F* which does the same thing as the command "ls -aF" if the shell variable *showdots* has been set, and acts like "ls -F" otherwise. Ls-F works like ls, only it is generally faster. If other switches are passed to *ls-F,* then the normal ls is executed. Aliasing ls to ls-F provides a fast alternative way of listing files. Note that on non BSD machines, where ls -C is not the default, *ls-F,* behaves like ls -CF.

ls-F appends the following characters depending on the file type:

=	File is an AF_UNIX domain socket. [if system supports sockets]
|	File is a named pipe (fifo) [if system supports named pipes]
%	File is a character device
#	File is a block device
/	File is a directory
*	File is executable

+	File is a hidden directory [aix] or context dependent [hpux]
:	File is network special [hpux]

On systems that support symbolic links the variable *listlinks* controls the way symbolic links are identified. If *listlinks* is not set then the character '@' is appended to the file. If *listlinks* is set then the following characters are appended to the filename depending on the type of file the symbolic links points to:

@	File is a symbolic link pointing to a non-directory
>	File is a symbolic link pointing to a directory
&	File is a symbolic link pointing to nowhere

While setting *listlinks* can be helpful while navigating around the filesystem, it slows down *ls-F* and it causes mounting of filesystems if the symbolic links point to an NFS automounted partition.

18. GLOBBING SYNTAX ADDITIONS

The syntax for any character in a range (for example "[a-z]*") has been extended so as to conform with standard Unix regular expression syntax (see *ed*(1)). Specifically, after an open bracket ("["), if the first character is a caret ("^") then the character matched will be any not in the range specified. For example:

```
> cd ~
> echo .[a-z]*
.cshrc .emacs .login .logout .menuwmrc
> echo .[^.]*
.Xdefaults .Xinit .cshrc .emacs .login .logout .menuwmrc
>
```

Note that the second form includes .Xdefaults and .Xinit because 'X' (and all the lower case letters) are outside of the range of a single '.'.

Also the ability to negate a globbing pattern has been added:

```
> echo *
foo foobar bar barfoo
> echo ^foo*
bar barfoo
```

Note that this does not work correctly if the expression does not have any wildcard characters (?*[]) or if the expression has braces {}.

19. NEW ENVIRONMENT AND SHELL VARIABLES

On startup, *tcsh* now automatically initializes the environment variable *HOST* to the name of the machine that it is running on. It does this by doing a *gethostname*(2) system call, and setting *HOST* to the result.

Tcsh also initializes the environment variable *HOSTTYPE* to a symbolic name for the type of computer that it is running on. This is useful when sharing a single physical directory between several types of machines (running NFS, for instance). For example, if the following is in *.login:*

```
set path = (~/bin.$HOSTTYPE /usr/ucb /bin /usr/bin /usr/games .)
```

and the user has directories named "bin.*machine*" (where *machine* is a name from the above list), then the user can have the same programs compiled for different machines in the appropriate "bin.*machine*" directories and *tcsh* will run the binary for the correct machine.

The current possible values are:

aix370	an IBM 370, running aix
alliant	an Alliant FX series
amdahl	an Amdahl running uts 2.1
apollo	an Apollo running DomainOS
att3b15	an AT&T 3b15
att3b2	an AT&T 3b2
att3b20	an AT&T 3b20
att3b5	an AT&T 3b5
balance	a Sequent Balance (32000 based)
butterfly	a BBN Computer Butterfly 1000
convex	a Convex
decstation	a DecStation XXXX
gould-np1	a Gould NP1
hk68	a Heurikon HK68 running Uniplus+ 5.0
hp300	an HP 9000, series 300, running mtXinu
hp800	an HP 9000, series 800, running mtXinu
hp9000s300	an HP 9000, series 300, running hpux
hp9000s500	an HP 9000, series 500, running hpux
hp9000s700	an HP 9000, series 700, running hpux
hp9000s800	an HP 9000, series 800, running hpux
hp	an HP, running hpux
i386	an Intel 386, generic
i386-mach	an Intel 386, running mach
intel386	an Intel 386, running INTEL's SVR3
iris3d	a Silicon Graphics Iris 3000
iris4d	a Silicon Graphics Iris 4D
isc386	an Intel 386, running ISC
m88k	an mc88000 CPU machine
mac2	an Apple Computer Macintosh II, running AUX
masscomp	a Concurrent (Masscomp), running RTU
mips	another mips CPU
multimax	an Encore Computer Corp. Multimax (32000 based)
news	a Sony NEWS 800 or 1700 workstation
news_mips	a NeWS machine with mips CPU
ns32000	an NS32000 CPU machine
next	a NeXT computer
pfa50	a PFU/Fujitsu A-xx computer
ps2	an IBM PS/2, running aix
ptx	a Sequent Symmetry running DYNIX/ptx (386/486 based)
pyramid	a Pyramid Technology computer (of any flavor)
rs6000	an IBM RS6000, running aix
rt	an IBM PC/RT, running BSD (AOS 4.3) or mach
rtpc	an IBM PC/RT, running aix
sco386	an Intel 386, running SCO
sun	a Sun workstation of none of the above types
sun2	a Sun Microsystems series 2 workstation (68010 based)
sun3	a Sun Microsystems series 3 workstation (68020 based)

sun386i	a Sun Microsystems 386i workstation (386 based)
sun4	a Sun Microsystems series 4 workstation (SPARC based)
symmetry	a Sequent Symmetry running DYNIX 3 (386/486 based)
titan	an Stardent Titan
unixpc	an UNIX/PC running SVR1 att7300 aka att3b1
vax	a Digital Equipment Corp. Vax (of any flavor)

(The names of the machines are usually trade marks of the corresponding companies.)

Tcsh also initializes the shell variables *uid* and *gid* to the value of the current real user ID/GID. This is useful for telling what user/group the shell is running as. Under Domain/OS *tcsh* will also set *oid* indicating the current real organization id.

20. COMMANDS FOR DEBUGGING

Only two such commands are available at this point, both concerned with testing termcap entries. *telltc* tells you, politely, what *tcsh* thinks of your terminal, and *settc* 'cap' 'value' tells *tcsh* to believe that the termcap capability 'cap' (as defined in *termcap*(5)) has the value 'value'. No checking for sanity is performed, so beware of improper use.

21. SEARCHING FOR THE VISUAL HISTORY

Two new editor functions have been added: history-search-backward, bound to M-p (and M-P), and history-search-forward, bound to M-n (and M-N). Each of these search backward (or forward) through the history list for previous (next) occurrence of the first word in the input buffer as a command. That is, if the user types:

```
> echo foo
foo
> ls
filea     fileb
> echo bar
bar
>
```

and then types "echo<ESC>p", the shell will place "echo bar" in the editing buffer. If another *M-p* was entered, the editing buffer would change to "echo foo". This capability is compatible with the plain visual history; if the user were to then enter *^P* the editing buffer would be changed to "ls". The pattern used to search through the history is defined by the characters from the beginning of the line up to the current cursor position and may contain a shell globbing pattern. Successive history searches use the same pattern.

22. BUILTIN WHICH(1) COMMAND

There is now a builtin version of the *which*(1) command. The builtin version is just like the original, except that it correctly reports aliases peculiar to this *tcsh,* and builtin commands. The only other difference is that the builtin runs somewhere between 10 and 100 times faster. There is also a key-function interface to this command: the *which-command* function (normally bound to M-?), can be used anywhere on the command line, and will in effect do a 'which' on the command name.

23. RESTARTING A STOPPED EDITOR

There is another new editor function: run-fg-editor, which is bound to *M-^Z.* When typed, it saves away the current input buffer, and looks for a stopped job with a name equal to the file name part (last element) of either the EDITOR or VISUAL environment variables (if not defined, the default names are "ed" and "vi" respectively). If such a job is found, then it is restarted as if "fg *%name*" had been typed. This is used to toggle back and forth between an editor and the shell easily. Some people bind this function to *^Z* so they can do this even more easily.

24. STATUS LINE SUPPORT

Tcsh has a new builtin called *echotc* that allows the user to access the terminal capabilities from the command line, similar to the system V *tput*(1).

```
> echotc home
```

Places the cursor at the home position and

```
> echotc cm 3 10
```

places the cursor at column 3 row 10. This command replaces the *el* and *sl* variables that used contain the escape sequences to begin and end status line changes. The command:

```
> echo $sl this is a test $el
```

is replaced by:

```
> echotc ts 0; echo "this is a test"; echotc fs
```

In addition *echotc* understands the arguments *baud, lines, cols, meta,* and *tabs* And prints the baud rate, the number of lines and columns, and "yes" or "no" depending if the terminal has tabs or a meta key. This can be useful in determining how terse the output of commands will be depending on the baud rate, or setting limits to commands like history to the highest number so that the terminal does not scroll:

```
> set history=`echotc lines`
> @ history--
```

Note: Termcap strings may contain wildcard characters, and echoing them will not work correctly. The suggested method of setting shell variables to terminal capability strings is using double quotes, as in the following example that places the date in the status line:

```
> set tosl="`echotc ts 0`"
> set frsl="`echotc fs`"
> echo -n "$tosl";date; echo -n "$frsl"
```

Echotc accepts two flags. The flag *-v* enables verbose messages and the flag *-s* ignores any errors and returns the empty string if the capability is not found.

25. EXECUTION OF A COMMAND AFTER CHANGING THE CURRENT WORKING DIRECTORY

Tcsh now supports a special alias, *cwdcmd,* which if set holds a command that will be executed after changing the value of $cwd. For example, if the user is running on an X window system *xterm(1),* and a reparenting window manager that supports title bars such as *twm(1) and has done:*

```
> alias cwdcmd  'echo -n "^[]2;${HOST}:$cwd ^G"'
```

then the shell will change the title of the running *xterm(1)* to be the name of the host, a colon, and the full current working directory. A more fancy way to do that is:

```
> alias cwdcmd 'echo -n "]2;${HOST}:$cwd]1;${HOST}"'
```

This will put the hostname and working directory on the title bar but only the hostname in the icon manager menu. Note that if a user defines *cwdcmd* to contain a *cd, pushd,* or *popd,* command, an infinite loop may result. In this case, it is the author's opinion that said user will get what he deserves.

26. NATIVE LANGUAGE SYSTEM

Tcsh is eight bit clean (if so compiled, see the description of the *version* shell variable below), and will thus support character sets needing this capability. The *tcsh* support for NLS differs depending on whether it was compiled to use the system's NLS (again, see the *version* variable) or not. In either case, the default for character classification (i.e. which characters are printable etc) and sorting is 7-bit ascii, and any setting or unsetting of the LANG or LC_CTYPE environment variables will cause a check for possible changes in these respects.

When using the system's NLS, the *setlocale* C library function will be called to determine appropriate character classification and sorting - this function will typically examine the LANG and LC_CTYPE variables for this purpose (refer to the system documentation for further details). Otherwise, NLS will be simulated, by assuming that the ISO 8859-1 character set is used whenever either of the LANG and LC_CTYPE variables are set, regardless of their values. Sorting is not affected for the simulated NLS.

In addition, with both real and simulated NLS, all printable characters in the range \200-\377, i.e. those that have M-<char> bindings, are automatically rebound to *self-insert-command* (the corresponding binding for the escape+<char> sequence, if any, is left alone). This automatic rebinding is inhibited if the NOREBIND environment variable is set - this may be useful for the simulated NLS, or a primitive real NLS which assumes full ISO 8859-1 (otherwise all M-<char> bindings in the range \240-\377 will effectively be undone in these cases). Explicitly rebinding the relevant keys, using *bindkey,* is of course still possible.

Unknown characters (i.e. those that are neither printable nor control characters) will be printed using the \nnn format. If the tty is not in 8 bit mode, other 8 bit characters will be printed by converting them to ascii and using standout mode. *Tcsh* will never change the 7/8 bit mode of the tty, and will track user-initiated settings for this - i.e. it may be necessary for NLS users (or, for that matter, those that want to use a Meta key) to explicitly set the tty in 8 bit mode through the appropriate *stty*(1) command in e.g. the .login file.

27. AUTOMATIC PROCESS TIME REPORTING

Automatic process time reporting is a feature that exists in *csh,* but it is usually not documented. In addition *tcsh* provides a slightly enriched syntax. Process time reports are controlled via the *time* shell variable. The first word of the *time* variable indicates the minimum number of CPU seconds the process has to consume before a time report is produced. The optional second word controls the format of the report. The following sequences are available for the format specification:

%U	The time the process spent in user mode in cpu seconds.
%S	The time the process spent in kernel mode in cpu seconds.
%E	The elapsed time in seconds.
%P	The CPU percentage computed as (%U + %S) / %E.

The following sequences are supported only in systems that have the BSD resource limit functions.

%W	Number of times the process was swapped.
%X	The average amount in (shared) text space used in Kbytes.

%D	The average amount in (unshared) data/stack space used in Kbytes.
%K	The total space used (%X + %D) in Kbytes.
%M	The maximum memory the process had in use at any time in Kbytes.
%F	The number of major page faults (page needed to be brought from disk).
%R	The number of minor page faults.
%I	The number of input operations.
%O	The number of output operations.
%r	The number of socket messages received.
%s	The number of socket messages sent.
%k	The number of signals received.
%w	Number of voluntary context switches (waits).
%c	Number of involuntary context switches.

The default time format is "%Uu %Ss %E %P %X+%Dk %I+%Oio %Fpf+%Ww" for systems that support resource usage reporting and "%Uu %Ss %E %P" for systems that do not.

For Sequent's DYNIX/ptx %X, %D, %K, %r and %s are not supported. However, the following additional sequences are available.

%Y	The number of system calls performed.
%Z	The number of pages which are zero-filled on demand.
%i	The number of times a process' resident set size was increased by the kernel.
%d	The number of times a process' resident set size was decreased by the kernel.
%l	The number of read system calls performed.
%m	The number of write system calls performed.
%p	the number of reads from raw disk devices.
%q	the number of writes to raw disk devices.

The default time format for Sequent's DYNIX/ptx is "%Uu %Ss $E %P %I+%Oio %Fpf+%Ww". Also note that the CPU percentage can be higher than 100% on multi-processors.

28. OS/DEPENDENT BUILTIN SUPPORT

TRANSPARENT COMPUTING FACILITY

On systems that support TCF (aix-ibm370, aix-ps2) the following builtins have been added:

getspath Print the current system execution path.

setspath LOCAL|<site>|<cpu> ...
Set the current execution path.

getxvers Print the current experimental version prefix.

setxvers [<string>]
If the optional string is ommited, any experimental version prefix is removed. Otherwise the experimental version prefix is set to string.

migrate [-<site>] <pid>|%<jobid> ...
migrate -<site>

The first form migrates the process or job to the site specified or the default site determined by the system path. The second form, is equivalent to 'migrate -<site> $$', i.e. migrates the current process to the site specified. Note: migrating *tcsh* itself can cause unexpected behavior, since the shell does not like to lose its tty.

In addition, jobs will print the site the job is executing.

Domain/OS Support

inlib <shared-library> ...
Inlib adds shared libraries to the current enviroment. There is no way to remove them...

rootnode //<nodename>
Change the name of the current rootnode. From now on, / will resolve to *//<rootnode>*

ver [<systype>] [<command>]
Without arguments, print *$SYSTYPE;* with the *<systype>* provided, set *SYSTYPE* to the one provided. Valid systypes are bsd4.3 and sys5.3. If a *<command>* is argument is given, then *<command>* is executed under the *<systype>* specified.

Mach

setpath <path-spec> ...
XXX: What does it do?

Masscomp/RTU

universe <universe-spec> ...
Sets the current universe to the specified parameter.

Convex/OS

warp [<universe-spec>] ...
Without arguments prints the current value of the universe. With a universe argument it sets the current universe to the value of the argument.

29. WINDOW SIZE TRACKING

On systems that support SIGWINCH or SIGWINDOW, *tcsh* adapts to window resizing automatically and adjusts the environment variables LINES and COLUMNS if set. Also if the environment variable TERMCAP contains li#, and co# fields, these will be adjusted also to reflect the new window size.

30. INPUT FILES

On startup *tcsh* will try to source */etc/csh.cshrc* and then */etc/csh.login* if the shell is a login shell. Then it will try to source *$HOME/.tcshrc* and then *$HOME/.cshrc* if *$HOME/.tcshrc* is not found. Then it will source *$HOME/.login* if the shell is a login shell. On exit *tcsh* will source first */etc/csh.logout* and then *$HOME/.logout* if the shell was a login shell.

Note: On *convexos* the names of the system default files are */etc/cshrc* , */etc/login* and */etc/logout* respectively and on *irix* or *A/UX* only the file */etc/cshrc* is executed if the shell is a login shell.

31. COMMAND LINE OPTIONS

This section describes options that are either undocumented in *csh* (*) or present only in *tcsh*. (+)

-d Load ~/.cshdirs (If *tcsh* was compiled with CSHDIRS enabled)(+)

-l Make *tcsh* behave like a login shell. (+)

-m Allow reading of a .cshrc that does not belong to the effective user. Newer versions of *su(1)* can pass that to the shell. (some versions of csh have it) (+*)

-q Make the shell accept SIGQUIT, and behave when it is used under a debugger. Job control is disabled. (*)

32. HISTORY AND VARIABLE MODIFIER ENHANCEMENTS

Tcsh accepts more than one variable modifier per variable or history expansion. For example, in *csh(1)* the following command expands to:

```
% set a=/usr/local/foo.bar.baz
% echo $a:t:r:e
foo.bar.baz:r:e
```

but in *tcsh:*

```
> set a=/usr/local/foo.bar.baz
> echo $a:t:r:e
bar
```

This bug fix changes slightly the input syntax of *csh,* causing expressions of the form to have invalid syntax:

```
> set a=/usr/local/foo.bar.baz
> echo $a:t:$cwd
Unknown variable modifier.
```

Which is the correct behavior, since after the second colon a variable modifier is expected and '$' is found. Expressions like this should be re-written as:

```
> echo ${a:t}:$cwd
```

FYI

This shell uses cbreak mode but takes typed-ahead characters anyway. You can still use *stty*(1) to set some of the modes of your terminal (but not bindings).

This shell will restore your tty to a sane mode if it appears to return from some command in raw, cbreak, or noecho mode. This behavior can be changed using *setty.*

ENVIRONMENT

HPATH -- path to look for command documentation
LANG -- used to give preferred character environment (see NLS)
LC_CTYPE -- used to change only ctype character handling (see NLS)
NOREBIND -- inhibits rebinding of printable characters to self-insert-command
PATH -- path to look for command execution
SHLVL -- current shell level nesting
TERM -- used to tell how to handle the terminal
LINES -- Number of lines in terminal (see WINDOW SIZE)
COLUMNS -- Number of columns in terminal (see WINDOW SIZE)
TERMCAP -- Terminal capability string (see WINDOW SIZE)
SYSTYPE -- The current system type (Domain OS only)

NEW SHELL VARIABLES

addsuffix add a / for directories, and a space for normal files when complete matches a name exactly. If unset don't add anything extra.

ampm show all times in 12 hour, AM/PM format.

autocorrect
Correct mis-spelled path components automatically before attempting completion.

autoexpand
invoke the expand-history function automatically on completion.

autolist list possibilities on an ambiguous completion.

autologout
number of minutes of inactivity before automatic logout.

backslash_quote
makes the backslash quote \, ', and ". This option changes the parsing mechanism for tcsh, and it can cause syntax errors in *csh* scripts.

chase_symlinks
always resolve symbolic links to real names on cd, etc.

correct automatically try to correct the spelling of commands. Must be set to either correct=cmd, only command name will be corrected, or correct=all, the whole line will be corrected.

dextract extract a directory on pushd rather than rotating.

edit use the input editor, set by default.

fignore list of file name suffixes (e.g. .o, ~) to ignore during complete.

gid the current real group id.

histlit If set, history lines in the editor will be shown with its literal value (that is the line as it was input) instead of the shells lexical version. The current history line can be toggled between literal and lexical with the toggle-literal-history function. History lines saved at shell exit are also saved as this variable indicates.

histfile If set, it contains the full path-name where a history file is read/written. It defaults to $home/.history. This is useful when sharing the same home directory in different machines, or if one wants to save all the histories in the tty sessions. It is usually set in .cshrc for interractive shells, because history is sourced between .cshrc and .login so that it is available from .login.

ignore_symlinks
don't resolve symbolic links to real names on cd, etc.

listjobs list all jobs when suspending. set listjobs=long, produces long format.

listlinks Resolve symbolic links when listing files so that the correct filetype is shown.

listmax maximum number of items to list without asking first.

matchbeep
control beeping on completion. With matchbeep=nomatch, completion only beeps when there is no match, with matchbeep=ambiguous, beeps also when there are multiple matches, with matchbeep=notunique, beeps when there is one exact and other longer matches, with matchbeep=never, it never beeps.

nobeep Disables beeping completely.

oid The organization id number (Domain OS only).

printexitvalue
if an interactive program exits non-zero, print the exit value.

prompt the string to prompt with.

prompt2 the string to prompt for while and for loops with.

prompt3 the string to prompt with when automatic spelling correction has corrected a command line.

pushdtohome
make pushd with no args do a "pushd ~" (like cd does).

pushdsilent
do not print the dir stack on every pushd and popd.

recexact recognize exact matches even if they are ambiguous.

recognize_only_executables
list choices of commands only displays files in the path that are executable (slow).

rmstar Prompt the user before execution of 'rm *'.

savehist number of history items to save between login sessions.

shlvl Integer value indicating the number of nested shells.

showdots show hidden files in list and complete operations.

tcsh Contains the current version of the shell as R.VV.PP. The *R* indicates the major release number, the *VV* the current version and the *PP* the patchlevel.

term the terminal type; see above.

tperiod periodic command wait period (in minutes).

tty The name of the tty, or empty if not attached to one.

uid the current real user ID.

version the version ID stamp for this *tcsh*. It contains, the origin of this version of *tcsh*, the date this version was released and a string containing a comma separated list of the compile time options enabled:

8b 7b	If tcsh was compiled to be eight bit clean or not. The default is 8b.
nls	Set if tcsh uses the system's NLS, should be the default for systems that have NLS.
lf	Set if tcsh should execute .login before .cshrc on login shells. Default is not set.
dl	Set if tcsh should put . last on the path for security. Default is set.
vi	Set if tcsh's default editor is vi. Default is unset (emacs)
dtr	Set if tcsh should drop dtr on login shells when exiting. Default is unset.
bye	Set if tcsh should accept bye in addition to logout, and rename log to watchlog. Default is unset.
al	Set if tcsh should determine if autologout should be enabled. The default is set.
dir	Set if tcsh should save and restore the directory stack.
kan	Set if tcsh is compiled for Kanji. (ignore the iso character set.) Default is unset.
sm	Set if tcsh was compiled to use the system's malloc.

In addition to the above strings, administrators can enter local strings to indicate differences in

the local version.

visiblebell
use the visible bell (screen flash) rather than audible bell.

watch list of events to watch.

who format string for the printouts generated when *watch* is set.

wordchars
list of nonalphanumeric characters considered part of a word for the purpose of the forward-word, backward-word etc functions -- defaults to "*?_-.[]~=".

NEW SPECIAL ALIASES

cwdcmd the command is run after every change of working directory.

periodic the command to be run every *tperiod* minutes.

precmd the command to be run prior to printing each prompt.

beepcmd the command to be run every time *tcsh* wants to echo the terminal bell.

SEE ALSO

xterm(1), twm(1), csh(1), chsh(1), termcap(5), termio(7)

BUGS

The screen update for lines longer than the screen width is very poor if the terminal cannot move the cursor up (i.e. terminal type "dumb").

VERSION

This man page documents tcsh 6.00.02 (Cornell) 08/05/91.

ConvexOS V10.0 Advance Notice
Part No. 710-010030-002