CoralCMD User Documentation

<*operz@coralcmd.net*>

`@(#)coralcmd.tr  1.3 2026/01/20 19:06:25 CORAL`

Welcome to CoralCMD.net! This is the user documentation for using our timesharing system's resources.

1. What is CoralCMD?

CoralCMD is a multi-user, timesharing system. In the modern age, these are often referred to as ‘‘pubnix’’ (for PUBlic uNIX) or ‘‘tilde’’ systems. These systems are services, where users may log in as a user, usually via a Unix shell over SSH (or, in less common cases, unencrypted telnet or rlogin). CoralCMD runs OmniOS, a distribution based on the Illumos operating system, which is actively developed as a fork of the now-defunct OpenSolaris OS/NET source code.

CoralCMD hosts several services for users to enjoy and use, including (but not limited to):

- Shell account access, which includes common development tools and version control for many programming languages and text editors, as well as thousands of commands for many different tasks
- Email accounts, from both mail clients on the shell and over the network via IMAP/POP3/SMTP
- Web space hosting, encrypted via HTTPS; this includes server-side CGI scripts
- Around 40GiB of disk space for each user's home directory
- (SOON TO COME) A PBX extension connected to the North American PSTN

Sign-up details and guidelines are located at: https://www.coralcmd.net/signup.html

2. Connecting over SSH to your shell account

TODO.

3. Tips on Unix shell usage

While you likely know a fair amount about Unix systems and usage of their shells already, here are some tips for lesser-known, yet extremely helpful things on our system:

(1) To change your shell, type `chsh`. To change your display name (also known as GECOS/Comment) field, type `chfn`. To change your password, type `chpass`. The `passwd` command has been deprecated and should no longer be used.

(2) While you may read the Unix manual pages with the `man` command, you may also use the `apropos` command to search for keywords within the man pages, in case you don't know the exact name. You may also suffix the page name with .NUMBER, where NUMBER is a manual section, such as `1` for user commands, or `3head` for headers of library functions. Example: `socket.3head`

(3) On CoralCMD, there are multiple versions of the same command installed in multiple locations, with some containing varying amounts of or different functionality. You may

view the full paths to these commands by using the `which -a` command. For example, for the `file` command: `which -a file`

(4) Editing your timezone to display correct dates is fairly simple. Append `export TZ=Region/Major_City` to your `$HOME/.profile` login script. An example of a city-based timezone would be `Australia/Perth`, and an example of a symbolic timezone would be `EST5EDT`. A list of valid timezones can be found at this link: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

(5) The `calendar` command and file format can be used to set up a reminder service. If you put "`12/17 17th of dec`" in `$HOME/calendar`, the calendar reminder service will send you an email at midnight UTC the night before and the day of the event.

(6) The `vacation` command can be used to set up an out-of-office message as an auto-reply to incoming emails. This can be set up via a user's `$HOME/.forward` file, which can also be used to forward emails to another user or address, a file, and pipe to a command (in this case, vacation). For details, see the manual pages vacation(1) and forward(5).

(7) The `quota -v` command will show how much disk space you have taken up and how much you have available.

4. Email

4.1. Email over the network

Protocol	Port	Encryption Type	Authentication Type
IMAP	993	SSL/TLS (Implicit Encryption)	PLAIN AUTH
POP3	995	SSL/TLS (Implicit Encryption)	PLAIN AUTH
SMTP	587	STARTTLS (Explicit Encryption)	PLAIN AUTH

4.2. Email from the command line

There are a few email clients installed that you may use on the command-line. Three notable ones are `mutt`, `alpine`, and `nmh`.

5. Manual Pages

You may view the manual pages using the `man` command as described in the tips section, but you may also list all the page titles and descriptions of a specific manual section, using this command:

```
apropos . | grep '(7)'
```

The `apropos` command usually lists the manual pages that match the keyword(s) given as command arguments, but in this case, the "." (dot) causes the command to list *all* of the manual pages, then the `grep` command sorts by the section.

Sometimes, there may be duplicate man page names, which causes confusion as the `man` command only shows the first result it finds. You may display your ‘:’ (colon) delimited MANPATH variable by typing , which you can then select a specific path to search for the manual page, by using this command (replace `MANPATH` with your location of choice):

```
MANPATH="/usr/share/man" man ksh
```

Or, you may search and display the results from all MANPATHs using the `manex` (MANual EXtended) script written by the administrators of CoralCMD to make things easier.

5.1. Manual Sections

(1)	User Commands
(1b)	BSD Compatibility Package Commands
(1c)	Communication Commands
(1s)	SunOS Specific Commands

(2)	System Calls

(3)	Introduction to Library Functions
(3bsm)	Security and Auditing Library Functions
(3c)	Standard C Library Functions
(3cfgadm)	Configuration Administration Library Functions
(3curses)	Curses Library Functions
(3devid)	Device ID Library Functions
(3devinfo)	Device Information Library Functions
(3elf)	ELF Library Functions
(3ext)	Extended Library Functions
(3gen)	String Pattern-Matching Library Functions
(3head)	Headers
(3kstat)	Kernel Statistics Library Functions
(3kvm)	Kernel VM Library Functions
(3ldap)	LDAP Library Functions
(3lib)	Interface Libraries
(3m)	Mathematical Library Functions
(3mail)	User Mailbox Library Functions
(3malloc)	Memory Allocation Library Functions
(3mp)	Multiple Precision Library Functions
(3nsl)	Networking Services Library Functions
(3pam)	PAM Library Functions
(3proc)	Process Control Library Functions
(3resolv)	Resolver Library Functions
(3rpc)	RPC Library Functions
(3sec)	File Access Control Library Functions
(3secdb)	Security Attributes Database Library Functions
(3socket)	Sockets Library Functions
(3volmgt)	Volume Management Library Functions
(3xcurses)	X/Open Curses Library Functions
(3xnet)	X/Open Networking Services Library Functions

(4)	Device and Network Interfaces
(4d)	Devices
(4fs)	File Systems
(4i)	Ioctl Requests
(4m)	STREAMS Modules
(4p)	Protocols

(5)	File Formats and Configurations

(7)	Standards, Environments, and Macros

(8)	Maintenance Commands and Procedures

(9)	Kernel Concepts
(9e)	Driver Entry Points
(9f)	Kernel Functions for Drivers
(9s)	Data Structures for Drivers

6. Editors and Shells

There are many editors installed on CoralCMD, to make you comfortable in whatever environment you want. Some notable examples include `emacs`, `vim`, `nvi`, `vi`, `nano`, and of course the line-oriented editors `ex`, `nex`, and `ed`.

We also have a selection of shells to choose from, including (but not limited to) `bash`, `ksh`, `csh`, and `tcsh`.

7. File and Directory Permissions

7.1. umask Command

The `umask` command is used to automatically set the `chmod` permission bits (which is three-to-four columns, with each value ranging from numbers 0 to 7) in either your current session, or, if you add it to your startup shell script, all sessions.

`umask` uses a *mask* to set its values, i.e. it uses subtraction from the highest down, so, to set `chmod 755`, you would run umask 022.

This is useful in case you want to change the default permissions for which you create files. By default your home directory and most (but not all, as some programs automatically restrict access to sensitive data such as mailboxes and SSH keys) files are *world-readable*, meaning any logged-in user on the system may view those files. If you do not like this, you may run the following commands:

```
find "$HOME" -type d -exec chmod 700 {} ;
find "$HOME" -type f -perm /111 -exec chmod 700 {} ;
find "$HOME" -type f ! -perm /111 -exec chmod 600 {} ;
```

And don't forget to add the appropriate `umask to your startup script:`

```
umask 0077
```

7.2. Unix permissions vs Extended (NFSv4/ZFS) ACLs

In most POSIX-conformant filesystems, you have your standard Unix-style permissions, modified with the `chmod` command. Usually this is in the syntax of something like `chmod ug+rwx filename.ext`. In ZFS, our filesystem of choice, we use NFSv4 extended ACLs as a way to supplement the existing POSIX-style ACLs with richer semantics, finer-grained access control and inheritance settings.

To use these extended ACLs, you may use the `chmod` command, but where the octal bits or symbolic letters appear, you create a long string of letters prefixed with `A+`, `A-`, or `A=`. An example would be:

```
chmod -R A+user:myusername:full_set:fd:allow file-or-directory-name
```

This gives a user full control in an ACE to not only read, write, execute, etc, but also to modify the ACLs to add more users or groups. You may also specify the symbolic letters by hand, as described in `chmod(1)`.

Below is an example of listing the ACLs on a file where users `europa` and `xm` have full access.

```
xm@coralcmd.net:…hare/acl-testing/europaTestDir$ ls -V
total 1
-rwxr--r--+  1 europa   other          0 May 25 20:15 europaTest
            user:europa:rwxpdDaARWc--s:------I:allow
                user:xm:rwxpdDaARWc--s:------I:allow
                 owner@:rwxpdDaARWc--s:------I:allow
                 owner@:rw-p--aARWcCos:-------:allow
                 group@:r-----a-R-c--s:-------:allow
              everyone@:r-----a-R-c--s:-------:allow
```

More info on how to work with extended ACLs (specifically NFSv4/ZFS style) may be found at the manual page chmod(1).

8. SVR4 (default) commands vs. BSD commands (secondary) vs. pkgsrc (supplementary)

W.I.P.

The default SVR4-style commands are located under /usr/bin and /usr/sbin, and the BSD-style commands are located under /usr/ucb. The BSD-style commands are appended near the end of all users' PATHs, for accessibility of commands that SVR4's style does not offer. pkgsrc's supplementary software commands are located under /opt/local/bin, and includes many programs, including TeXlive and ImageMagick.

9. User Service Management For managing background services as a user, `dinit` has been added to the system. On boot, all interactive users (that means you, too!) have a process started which allows services to be run in the background. If you would like to run a process in the background 24/7, such as a network listener or an unprivileged daemon, you may do so by creating a service descriptor, like so, inside `~/.config/dinit.d/servicename`:

```
type = process
command = /path/to/executable --arguments one two etc
restart = true
```

Then, you may enable the service and start it by running:

```
$ ln -s ~/.config/dinit.d/servicename ~/.config/dinit.d/boot.d/servicename
$ dinitctl start servicename
```

You may check the status of services by running `dinitctl list`. More information may be found in the manual pages for dinit, dinitctl, and at the dinit homepage at `https://davmac.org/projects/dinit/`.

10. Websites & CGI (Common Gateway Interface)

Each user has their own website, located publicly at https://username.coralcmd.net/, and locally accessible via their own home directory under `~/public_html`.

By default, each hour, permissions for the web directory are reset to a sane default to prevent issues with the web server accessing the files.

10.1. .htaccess File

Placing a file named `.htaccess` in your public HTML directory or any subdirectory will have the web server parse it for different directives, including (but not limited to) password protection, file and directory permissions, and SSL/TLS enforcement.

10.1.1. Example: Forcing SSL/TLS (HTTPS) Mode

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
```

10.2. Common Gateway Interface

This document will not go into the actual programming aspect of CGI, however the way you may set it up is via an exectuable (chmod +x) file ending with the extension `.cgi`. Once that file is created, the standard output must conform to the CGI standard, which, essentially, is just a glorified HTTP protocol output. The actual executable file format can be one of two types: an exectuable binary (ex. a C program compiled into an ELF binary) or a script using an interpreter such as Python 3 or Korn Shell or Perl.

The script style must, on the first line, contain a ‘‘shebang’’ and then the full system path to the interpreter. As mentioned before, the full system path to an executable binary can be found via the command `which`. An example for a bash script is shown below:

```
#!/usr/bin/bash
printf "Status: 200 OK\r\n"
printf "Content-Type: text/plain\r\n"
printf "\r\n"
printf "hello world\r\n"
```

Notice that according to the CGI and HTTP standards, each line must end with both a carriage return and line feed character (CRLF).