NOTES

NOTES on minishell commands

Readline

- Readline info link
- Includes:

```
#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>
```

- On mac compile with: -lreadline
- Functions:

1. **readline**: `char *readline(const char *prompt)`
    - if prompt is NULL, no prompt is used.
    - Needs free coz it's malloc'd
2. **rl_clear_history**: `void rl_clear_history(void)`
    - Clear the history by deleting all the entries. It frees private data Readline saves in the history list.
3. **rl_on_new_line**: `int rl_one_new_line(void)`
    - Tell the update functions (like rl_redisplay) we have moved onto a new (empty) line, usually after outputting a newline.
4. **re_redisplay**: `void rl_redisplay(void)`
    - Change what's displayed on the screen to reflect the current contents of rl_line_buffer(in our case, should be the buffer we manipulate).

FIles & Directories

- General Info
- The link above contains possible info & errors explanations in detail.
- Rules about file descriptors apply to directory streams as well
- In most cases below, *errno* set to indicate the error.

1. **getcwd**:
    - get the pathname of the current work directory

```c
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

2. **stat**:
   - the stat() shall obtain info about the named file and write it to the area pointed to by the *buf* argument. The *path* arg points to a pathname naming a file.
   - Return value : upon success: 0 -- else: -1

```c
#include <sys/stat.h>

int stat(const char *restrict path,
struct stat *restrict buf);
```

3. **lstat**:
   - Equivalent to stat(), except when path refers to symbolic link. In that case lstat() shall return info about the link.
   - Return value and prototype are the same as in stat().
4. **fstat**:
   - The fstat() shall obtain info about an open file associated with the file descriptor fd and shall write it to the area pointed to by buf.
   - Return value : upon success: 0 -- else: -1

```c
#include <sys/stat.h>

int fstat(int fd, struct
stat *buf);
```

5. **chdir**:
   - The chdir() changes the cwd to *path*, which can be relative to the cwd or an absolute path name.
   - Return Value: upon success: 0 -- else: -1

```c
#include <unistd.h>

int chdir(const char *path);
```

6. **opendir**
   - The opendir() function shall open a directory stream corresponding to the directory named by the *dirname* argument. The directory stream is positioned at the first entry. If the type DIR is implemented using a file descriptor, applications shall only be able to open up to a total of {OPEN_MAX} files and directories.

- Return value: upon success: a pointer to an object of type DIR. -- else: null pointer.

```c
#include <dirent.h>

DIR *opendir(const char *dirname);
```

7. **readdir**:
   - Read directory
   - Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of readdir().
   - Readdir() returns a pointer to a struct representing the dir. entry at the current position in the dir. stream specified by the argument *dirp*, and positions the dir stream at the next entry. It returns a null pointer upon reaching the end of the dir. stream. The structure *dirent* defined by <dirent.h> describes a dir. entry.
   - After (and if) fork() happens, either the parent or the child (but not both) may continue processing the directory stream using readdir().

```c
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
```

8. **closedir**
   - The closedir() closes the dir. stream associated with *dirp*. A successful call to closedir() also closes the underlying file descriptor associated with *dirp* and makes the stream unavailable (obviously).
   - Return Value: upon success: 0 -- else: -1

```c
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR *dirp);
```

Terminal

- When opening a terminal device with the O_NONBLOCK flag clear shall cause the thread to block until the terminal device is ready and available.
- In most cases below, *errno* set to indicate the error.
- Includes:

```c
#include <unistd.h>
#include <stdlib.h>
```

1. **isatty**:

- This function shall test whether *fd*, an open file descriptor, is associated with a terminal device.
- Return Value: upon success: 1 -- else: 0
- `int isatty(int fd);`

2. **ttyname**:

   - This function returns a string with the pathname of the terminal associated with the *fd*.
   - The return value may point to static data whose content is overwritten by each call.
   - Return value: upon success: a pointer to a string -- else: null pointer
   - `char *ttyname(int fd);`

3. **ttyslot**:

   - This function returns the index of the current user's entry in the user acoounting database.
   - Troll function, avoid it.
   - More info (not for the lighthearted)
   - `int ttyslot(void);`

4. **ioctl**:

   - This function shall perform a variety of control functions on STREMS devices. The *request* argument and an optional third argument (with varuing type) shall be passed to and interpreted by the appropriate part of the STREAM associated with *fd*.
   - The *fd* argument is an open file descriptor that refers to a device.
   - The *request* argument selects the control function to be performed and shall depend on the STREAMS device being addressed.
   - The *arg* argument represents additional information that is needed by this specific STREAMS device to perform the requested function. The type of *arg* depends upon the particular control request, but it shall be either an integer or a pointer to a device specific data structure.
   - Return value: upon success: 0 --else: -1
   - For more info on ioctl() commands/error: go there

   ```
   #include <sys/ioctl.h>

   int ioctl(int fd, int request, ...);
   ```

5. **getenv**:

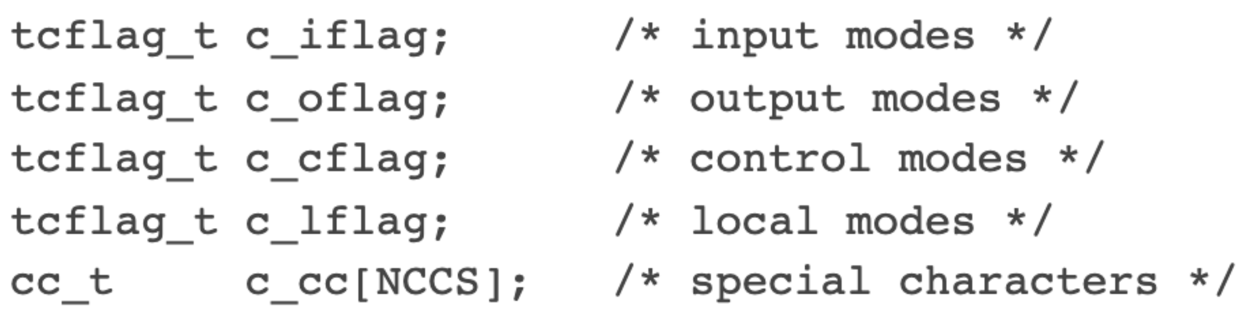
   - This function will search the environment of the calling process for the environmental variable *name* if it exists and return a pointer to the value of the environment variable.
   - The application shall ensure that it does not modify the string pointed to by the getnev() function.
   - It need not be reentrant.

   - Return value: upon success: a pointer to a string containing the value for the specified name -- else: null pointer.

   ```
   #include <stdlib.h>

   char *getenv(char const *name);
   ```

6. **tcgetattr & tcsetattr**:
   - As usual, *errno* set to indicate the error
   - The termios functions describe a general terminal interface that is provided to control asynchronous communications ports.
   - *just for reference*: The *termios_p* argument refers to a *termios* struct, which contains at least the following members:

     ```
     tcflag_t c_iflag;      /* input modes */
     tcflag_t c_oflag;      /* output modes */
     tcflag_t c_cflag;      /* control modes */
     tcflag_t c_lflag;      /* local modes */
     cc_t     c_cc[NCCS];   /* special characters */
     ```

   - We can set a terminal's attributes by using *tcgetattr()* to read the current attributes into a struct, modifying the struct by hand and passing the modified struct to *tcsetattr()* to write a new terminal attributes back out.
   - If the terminal device supports different input and output baud rates, the baud rates stored in the termios structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are equal.
   - Return value: upon success: 0 -- else -1.

   ```
   #include <termios.h>
   int tcsetattr(int fd, int optional_actions,
   const struct termios *termios_p);
   int tcgetattr(int fd, struct termios *termios_p)
   ```

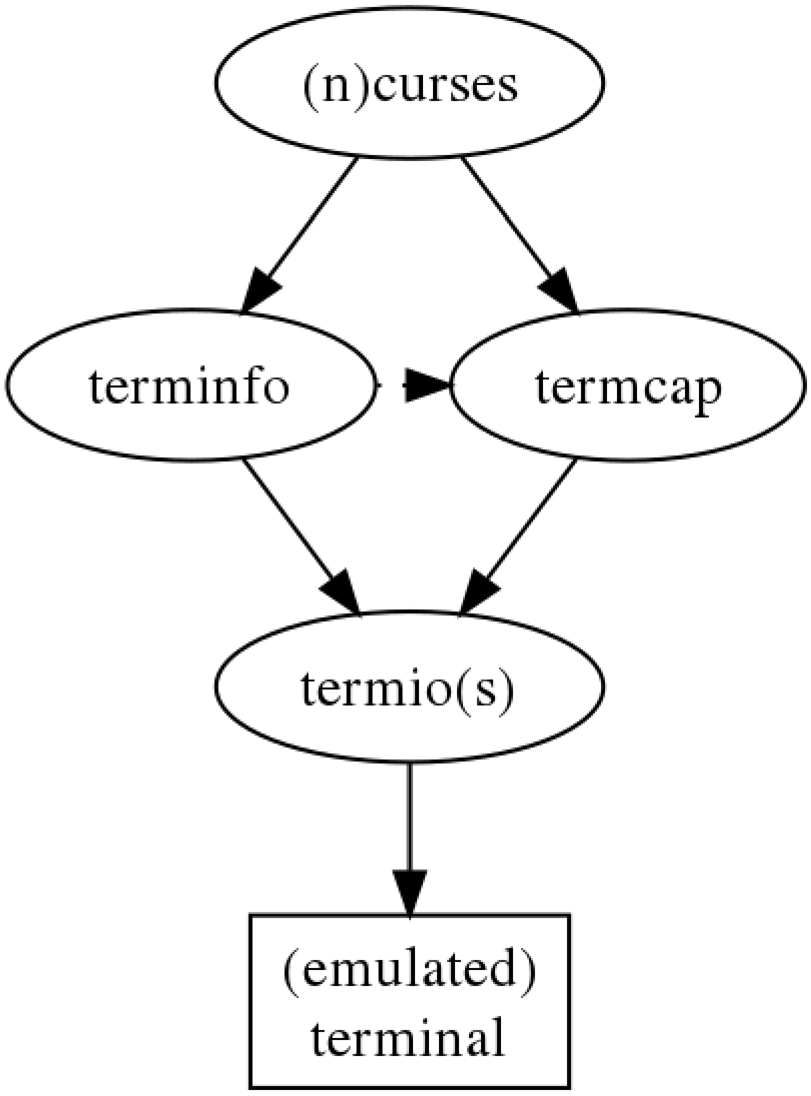
7. **tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs**
   - (from manual) direct **curses** interface to the terminfo capability database
   - **Terminfo**: terminal capability database. It is a database describing terminals used by screen-oriented programs such as nvi, rogue and libraries such as **curses**.
   - **curses**: is a terminal control library for UNIX-like systems, enabling the construction of text user interface (TUI) applications, without writing directly for any specific terminal type.

   ```
   #include <curses.h>
   #include <term.h>

   ```

```c
int tgetent(char *bp, const char *name);
// loads the entry for name
int tgetflag(char *id);
// gets the boolean entry for id
int tgetnum(char *id);
// gets the numeric entry for id
char *tgetstr(char *id, char **area);
// returns the string entry for id, use tputs to output the string
char *tgoto(const char *cap), int col, int row);
// instantiates the parameters into the given capabiility
int tputs(const char *str, int affcnt, int (*putc)(int));
// it can retrieve capabilitites by the respective database
```

- **TL;DR** This is how we configure our terminal.


(n)curses
terminfo
termcap
termio(s)
(emulated) terminal