**Week 08 – Terminal handling**

**Terminals**

* a dumb terminal is basically a text display with a keyboard, similar to an old teletype terminal
* companies started making dumb terminals "smarter", with cursor positioning and graphics
* special sequences of characters for special functions, such as cursor keys ("control sequences", "escape sequences")
* different terminal types have different sequences, no standards
* example VT100 escape sequences:
* move the cursor to row 12, column 53: ESC[12;53H
* clear the screen: ESC[HESC[J
* cursor up: ESC[A
* real dumb terminals are largely a thing of the past, instead we emulate their function with programs such as telnet, ssh, putty
* Unix is very good at handling many different kinds of terminals
* uses a standard terminal interface called "termios" to set terminal characteristics
* uses a standardized terminal description database called "terminfo" to use terminal escape sequences

**termios - set terminal characteristics**

* "termios" controls behaviour of the Unix terminal driver, including:
  + echoing on/off
  + input buffered until newline, or able to read each character immediately
  + interpretation of backspace: edit or read directly
  + character conversions (CR/LF)
* in shell scripts you use the "stty" command
  + stty                    - will show an abbreviated list of terminal capabilities
  + stty -a                 - will show all terminal capabilities
  + stty -g                 - will show all terminal capabilities in stty-readable form
* can be used to reset terminal characteristics
  + stty -echo              - disables echoing of typed characters
  + stty -icanon min nnn     - turns off canonical mode, doesn't require "enter" to read
    - "nnn" indicates number of characters accepted
  + stty -icrnl              - disables carriage-return to newline conversion
  + stty echo icanon icrnl  - enables these capabilities
* if you get into problems with terminal control, try typing "stty sane" or "control-j stty sane control-j"
  + try:   man termios   and   man stty

**Examples for termios**

* an example of a simple "password" script:

echo -n "Enter password: "stty -echoread passwordstty echoecho -e "\nYou entered '$password'"

* the "password" script using the $REPLY variable instead of a "read" variable:

echo -n "Enter password: "stty -echoreadstty echoecho -e "\nYou entered '$REPLY'"

* the "password" script using the "-s" (secure) and "-p" (prompt) option of "read" (bash only):

read -s -p "Enter password: "echo -e "\nYou entered '$REPLY'"

* an example to read one character at a time (without requiring "enter"):

stty -icanon min 0key=while [ "$key" != "q" ]doecho -n "Hit any key: "key=while [ "$key" = "" ]doread keydoneecho -e "\nYou hit the \"$key\" key"donestty icanon

* - try entering "\"
* - "-r" (raw) "read" option can be used so that "\" is handled like any other character
* an improved example, avoiding the "read" loop:

oldsettings=$(stty -g)stty -icanon min 1key=while [ "$key" != "q" ]doecho -n "Hit any key: "key=$(dd bs=1 count=1 2> /dev/null)echo -e "\nYou hit the \"$key\" key"donestty $oldsettings

* - try the "Up", "Down", "Left", "Right", and "Enter" keys
* improvements in this example:
* original terminal settings are saved, and later restored
* minimum keystrokes accepted has been changed from 0 to 1, to avoid the use of the "read" loop
* "dd" is used to read from the keyboard, instead of "read"
* "read" has been designed for canonical mode, hence the need for a loop when using non-canonical mode
* the "dd" blocksize has been set to 1, to read one character at a time
* the "dd" block count has been set to 1
* another improved example, using a bash-shell-only option:

key=while [ "$key" != "q" ]doecho -n "Hit any key: "key=read -n 1 -r keyecho -e "\nYou hit the \"$key\" key"done

* improvement in this example (bash only):
* "-n 1" option used to read one key at a time, the "read" loop is not needed
* again, "-r" (raw) "read" option can be used so that "\" is handled like any other character
* an example of a "pause" type of function for use within a script:

pause () {oldsettings=$(stty -g)stty -echo -icanon min 1echo -n "Press any key to continue..."key=$(dd bs=5 count=1 2> /dev/null)echostty $oldsettings}pause

* - use "bs=5" to account for multiple-character keys, such as "F1"
* an example of a "confirmation" type of function for use within a script:

confirm() {while read -r -n 1 -s -p "Please confirm (Y/N): " replydoecho[[ $reply = [Yy] ]] && return 0[[ $reply = [Nn] ]] && return 1done}if confirmthenecho "Okay, ..."elseecho "Operation cancelled"fi

* - note that this is a "bash only" version
* - the extended tests are using globbing comparisons, not regular expressions
* - try entering a function key, such as "F8"
* a similar "confirmation" function for use within a script, solving the multiple-character-per-key problem:

confirm() {stty -echo -icanon min 1while echo -n "Please confirm (Y/N): ";reply=$(dd bs=5 count=1 2> /dev/null)doecho[[ $reply = [Yy] ]] && { stty echo icanon; return 0; }[[ $reply = [Nn] ]] && { stty echo icanon; return 1; }done}if confirmthenecho "Okay, ..."elseecho "Operation cancelled"fi

**terminfo - use terminal escape sequences**

* terminals are listed by name under lettered directories in /usr/share/terminfo
* three types of capabilities
* boolean (e.g. "os" - overstrike is supported)
* numeric (e.g. "cols#80" - number of columns is 80)
* string (e.g. "cup=..." - escape sequence to set cursor position)
* terminfo files are binary, compiled using the "tic" command
* use command "infocmp" to de-compile, show original text entry
* TERM environment variable selects which terminfo entry to use for the shell, editors, etc.
* if "clear" command doesn't work then your TERM setting is definitely wrong!

echo $TERMTERM=vt100

* example of terminfo entry, using infocmp vt100:

vt100|vt100-am|dec vt100 (w/advanced video),am, mc5i, msgr, xenl, xon,cols#80, it#8, lines#24, vt#3,acsc=``aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,bel=^G, blink=\E[5m$&2>, bold=\E[1m$<2>,clear=\E[H\E[J$<50>, cr=^M, csr=\E[%i%p1%d;%p2%dr,cub=\E[%p1%dD, cub1=^H, cud=\E[%p1%dB, cud1=^J,cuf=\E[%p1%dC, cuf1=\E[C$<2>,cup=\E[%i%p1%d;%p2%dH$<5>, cuu=\E[%p1%dA,cuu1=\E[A$<2>, ed=\E[J$<50>, el=\E[K$<3>, el1=\E[1K$<3>,enacs=\E(B\E)0, home=\E[H, ht=^I, hts=\EH, ind=^J, ka1=\EOq,ka3=\EOs, kb2=\EOr, kbs=^H, kc1=\EOp, kc3=\EOn, kcub1=\EOD,kcud1=\EOB, kcuf1=\EOC, kcuu1=\EOA, kent=\EOM, kf0=\EOy,kf1=\EOP, kf10=\EOx, kf2=\EOQ, kf3=\EOR, kf4=\EOS, kf5=\EOt,kf6=\EOu, kf7=\EOv, kf8=\EOl, kf9=\EOw, lf1=pf1, lf2=pf2,lf3=pf3, lf4=pf4, mc0=\E[0i, mc4=\E[4i, mc5=\E[5i, rc=\E8,rev=\E[7m$<2>, ri=\EM$<5>, rmacs=^O, rmam=\E[?7l,rmkx=\E[?1l\E>, rmso=\E[m$<2>, rmul=\E[m$<2>,rs2=\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h, sc=\E7,sgr=\E[0%?%p1%p6%|%t;1%;%?%p2%t;4%;%?%p1%p3%|%t;7%;%?%p4%t;5%;m%?%p9%t\016%e\017%;$<2>,sgr0=\E[m\017$<2>, smacs=^N, smam=\E[?7h, smkx=\E[?1h\E=,smso=\E[7m$<2>, smul=\E[4m$<2>, tbc=\E[3g,

* tput queries and uses terminal capabilities
* tput cols        - gives number of columns in current display
* tput lines       - gives number of lines in current display
* tput cup 12 53   - move cursor position to row 12 column 53
* tput smso        - set mode standout (highlighting)
* tput rmso        - remove mode standout
* tput smul        - set mode underlining
* tput rmul        - remove mode underlining
* tput rev         - set reverse video, similar to standout with some terminal types
* tput sgr0        - remove all attributes
* tput sc          - save cursor position
* tput rc          - restore cursor position
* tput kbs         - backspace character sent by terminal
* tput cub 3       - move cursor back 3 characters (preceded by 'k' when sent by terminal)
* tput cuf 5       - move cursor forward 5 characters
* tput cuu 2       - move cursor up two lines
* tput cud 4       - move cursor down four lines
* tput el1         - clear to beginning of line
* tput el          - clear to end of line
* tput ed          - clear to end of screen
* man pages available for "terminfo", "tic", "infocmp", and "tput"

**Examples for terminfo**

* a simple script to play with most of the terminal capabilities noted above:

while truedoread -r -p "Enter tput arguments: "tput $REPLYdone

* an example using cursor positioning and highlighting:

string=while [ "$string" != 'q' ]doclearif [ "$string" != "" ]then tput cup 20 10tput smsoecho -n "You entered \"$string\""tput rmsofitput cup 10 10echo -n "Enter something: "read stringdoneclear

* another example, using highlighting and underlining within a line:

bold=$(tput smso)boldoff=$(tput rmso)ul=$(tput smul)uloff=$(tput rmul)echo "${bold}This${boldoff} should be highlighted and ${ul}this${uloff} should be underlined."

* another example, to read one keystroke, including cursor keys:

oldsettings=$(stty -g)stty -icanon min 1 -icrnl -echokey=tput smkx # set "keypad send mode", needed for cursor keyswhile [ "$key" != "q" ]doecho -n "Hit any key: "key=$(dd bs=3 count=1 2> /dev/null)if [ "$key" = $(tput cr) ]then key="Enter"elif [ "$key" = $(tput kcuu1) ]then key="Up"elif [ "$key" = $(tput kcud1) ]then key="Down"elif [ "$key" = $(tput kcub1) ]then key="Left"elif [ "$key" = $(tput kcuf1) ]then key="Right"elif [ "$key" = $(tput kbs) ]then key="Backspace"fiecho -e "\nYou hit the \"$key\" key"donestty $oldsettings

* note the following:
* CR/NL conversion has been disabled, so "Enter" key can be handled
* echoing has been turned off, so cursor keys don't display control sequences
* "tput smkx" used, so cursor keys can be handled
* "dd" blocksize has been set to 3, because the cursor keys have control sequences consisting of 3 characters
* "Enter" and cursor keys have been translated for readability

**case**

* case control structure, similar to "switch" in C
* simplifies a common "if-else" structure
* here's an example of "case" within a script:

echo -n "Enter the name of an animal: "read animalecho "Here are some interesting facts about ${animal}s:"case $animal inlion) echo "Baby lions are cute"echo "Lions are generally scaredy-cats";;tiger) echo "Tigers have stripes"echo "Tigers are native to Detroit";;bear) echo "Oh my!!!"echo "Bears are big and hungry and generally not sociable";;\*) [[ $animal =~ ^[aeiou] ]] && n=n || n=echo "I don't know what a$n $animal is, but I'm sure it's awesome!";;esac

* here's an example of a "case" variable using command substitution:

for filedoif [ ! -e "$file" ]thenecho "$file does not exist"elsecase $(ls -ld "$file" | cut -c1) in-) echo "$file is an ordinary file" ;;d) echo "$file is a directory" ;;l) echo "$file is a symbolic link" ;;[bc]) echo "$file is a device driver" ;;p) echo "$file is a pipe" ;;s) echo "$file is a socket" ;;esacfidone

**select**

* select control structure, designed to simplify creating a selection menu
* here's an example of "select" within a script:

select animal in lion tiger beardoecho "You selected $animal"breakdone

* an example of selecting from a list of files:

PS3="Please select a file: "select file in \*dols -ld $filebreakdone

* an example using case and select:

case $# in0) ;;1) if [ -d "$1" ]thencd "$1"elseecho "$1 is not a valid directory name" >&2exit 1fi ;;\*) echo "Syntax: $(basename $0) [ dir-name ]" >&2exit 2 ;;esacPS3="Please choose a file: "quit=while [ "$quit" != q ]doclearecho -e "Directory $PWD\n"select file in .\* \*doif [ -d "$file" ]thencd "$file"elsels -ld "$file"read -p "Hit enter to continue ('q' to quit): " quitfibreakdonedone

* note that basename removes any leading directory components from a pathname, if there are any
* similarly, dirname displays only leading directory components from a pathname, or '.' if there are none
* similarly, realpath displays absolute path of a pathname
* the same example without case and select:

if [ $# = 1 ]then if [ -d "$1" ]thencd "$1"elseecho "$1 is not a valid directory name" >&2exit 1fielif [ $# -gt 1 ]thenecho "Syntax: $(basename $0) [ dir-name ]" >&2exit 2fiquit=while [ "$quit" != q ]doclearecho -e "Directory $PWD\n"ls -a | awk '{print NR ") " $0}'read -p "Please choose a file: " filenumfile=$(ls -a | sed -n "$filenum p")if [ -d "$file" ]thencd "$file"elsels -ld "$file"read -p "Hit enter to continue ('q' to quit): " quitfidone