فایل منیجر ترمینال لینوکس

رحمت اله انصاری

۹۹۱۲۳۷۷۳۳۱

درس آزمایشگاه سیستم عامل

استاد دشتی

دانشگاه حکیم سبزواری

رشته مهندسی کامپیوتر

دی ۱۴۰۱

بسم الله الرحمن الرحیم

مشخصات

عنوان پروژه : فایل منیجر ترمینال لینوکس

نویسنده:

نام و نام خانوادگی : رحمت اله انصاری

شماره دانشجویی : 9912377331

ایمیل : Rahmat2022a@gmail.com

گیت‌هاب: github.com/EnAnsari

درس آزمایشگاه سیستم عامل

استاد : مهندس دشتی

دانشگاه حکیم سبزواری، سبزوار

رشته مهندسی کامپیوتر

دانشکده مهندسی کامپیوتر و برق

نوشته شده در دی سال ۱۴۰۱

فهرست مطالب

رنگ‌ها در شل 4
عملیات‌های قابل انجام در فایل منیجر 6
توضیح کد 7

رنگ‌ها در شل

برای تغییر رنگ به صورت زیر از پسوندهای پس از اسلش استفاده می‌کنیم:

```
# Reset
Color_Off='\033[0m'     # Text Reset

# Regular Colors
Black='\033[0;30m'      # Black
Red='\033[0;31m'        # Red
Green='\033[0;32m'      # Green
Yellow='\033[0;33m'     # Yellow
Blue='\033[0;34m'       # Blue
Purple='\033[0;35m'     # Purple
Cyan='\033[0;36m'       # Cyan
White='\033[0;37m'      # White

# Bold
BBlack='\033[1;30m'     # Black
BRed='\033[1;31m'       # Red
BGreen='\033[1;32m'     # Green
BYellow='\033[1;33m'    # Yellow
BBlue='\033[1;34m'      # Blue
BPurple='\033[1;35m'    # Purple
BCyan='\033[1;36m'      # Cyan
BWhite='\033[1;37m'     # White

# Underline
UBlack='\033[4;30m'     # Black
URed='\033[4;31m'       # Red
UGreen='\033[4;32m'     # Green
UYellow='\033[4;33m'    # Yellow
UBlue='\033[4;34m'      # Blue
UPurple='\033[4;35m'    # Purple
UCyan='\033[4;36m'      # Cyan
UWhite='\033[4;37m'     # White

# Background
On_Black='\033[40m'     # Black
On_Red='\033[41m'       # Red
On_Green='\033[42m'     # Green
On_Yellow='\033[43m'    # Yellow
On_Blue='\033[44m'      # Blue
On_Purple='\033[45m'    # Purple
On_Cyan='\033[46m'      # Cyan
On_White='\033[47m'     # White

# High Intensity
IBlack='\033[0;90m'     # Black
IRed='\033[0;91m'       # Red
IGreen='\033[0;92m'     # Green
IYellow='\033[0;93m'    # Yellow
IBlue='\033[0;94m'      # Blue
```

```
IPurple='\033[0;95m'     # Purple
ICyan='\033[0;96m'       # Cyan
IWhite='\033[0;97m'      # White

# Bold High Intensity
BIBlack='\033[1;90m'     # Black
BIRed='\033[1;91m'       # Red
BIGreen='\033[1;92m'     # Green
BIYellow='\033[1;93m'    # Yellow
BIBlue='\033[1;94m'      # Blue
BIPurple='\033[1;95m'    # Purple
BICyan='\033[1;96m'      # Cyan
BIWhite='\033[1;97m'     # White

# High Intensity backgrounds
On_IBlack='\033[0;100m'   # Black
On_IRed='\033[0;101m'     # Red
On_IGreen='\033[0;102m'   # Green
On_IYellow='\033[0;103m'  # Yellow
On_IBlue='\033[0;104m'    # Blue
On_IPurple='\033[0;105m'  # Purple
On_ICyan='\033[0;106m'    # Cyan
On_IWhite='\033[0;107m'   # White
```

رنگ‌ها به صورت بالا استفاده می‌شود. مثلا اگر بخواهیم به رنگ قرمز با زیر خط متن `hello world` بنویسیم. برای اینکار می‌توانیم به صورت زیر عمل کنیم:

```
$ printf '\033[4;31mRahmat FML\033[0m\n'
```

همچنین می‌توانیم از این مقادیر در مبناهای مختلف استفاده کنیم:

	bash	hex	octal	note
Start	\e	\x1b	\033	
start	\E	\x1B	-	x cannot be capital
End	\e[0m	\x1b[0m	\0333[0m	
end	\e[m	\x1b[m	\0333[m	0 is appended if you omit it

عملیات‌های قابل انجام در فایل منیجر

عملیات‌های کنترلی

کامند	:
نمایش فایل‌های پنهان	A
ساخت دایرکتوری	D
ساخت فایل	F
حذف فایل یا دایرکتوری نشان شده	X
خروج	Q

انتقال‌ها

عقب	H, ←
انتقال به پایین	J, ↓
انتقال به بالا	K, ↑
ورود به فایل یا دایرکتوری	L, →, ⏎

توضیح کد

ساخت سوییچ سازنده برای نوشتن مشخصات نویسنده:

```
if [[ $1 = "--writer" ]]
then
    echo '[F]ile [M]anager for [L]inux'
    echo 'Writer: Rahmatollah Ansari'
    echo 'Student Code: 9912377331'
    echo 'github: github.com/EnAnsari'
    echo 'Email: Rahmat2022a@gmail.com'
    echo ''
    exit 0
fi
```

برخی تنظیمات برای ساخت خروجی درست

```
set -eEuo pipefail ignoreeof
stty -echoctl
LC_ALL=C
LANG=C
```

دستور `stty` در لینوکس برای تغییر و چاپ تنظیمات خط ترمینال استفاده می شود. اساساً این دستور ویژگی های ترمینال را نشان می دهد یا تغییر می دهد. مثلا ویژگی های ترمینال را نمایش می دهد.

`Ignoreeof` چیست؟

یک پوسته تعاملی با خواندن `EOF` خارج نمی شود.

`Pipefail` چیست؟

اگر تنظیم شود، مقدار بازگشتی یک خط لوله، مقدار آخرین (راست ترین) دستور برای خروج با وضعیت غیر صفر یا صفر است اگر همه دستورات خط لوله با موفقیت خارج شوند. این گزینه به طور پیش فرض غیر فعال است.

برای نمایش یک ترمینال که کرسر چشمک زن نداشته باشد و محیطی شبیه خروجی بسازد از کد زیر استفاده می‌کنیم:

```
init_term() {
    shopt -s checkwinsize; (:;:) && ((rows=LINES-1))
    printf '\e[?1049h\e[2J\e[?7l\e[?25l\e[1;%dr\e[%dH' "$LINES" "$rows"
}
```

قبل از خروج هم تابع زیر اجرا می‌شود:

```
end() {
    printf '\e[?1049l\e[2J\e[?7h\e[?25h' && clear && exit
}
```

بعد از دبل امپرسنت تنها در صورتی اجرا می‌شود که قبل از آن درست اجرا شده باشد و به بیانی دیگر خروجی صفر داشته باشد. دستور `clear` ترمینال را پاک می‌کند و دستور `exit` برای خروج از ترمینال است. ما برای اجرای این برنامه یک ترمینال جدید باز کردیم. (امکان عدم ساپورت در برخی از لینوکس‌ها و ورژن‌هایشان)

از `printf` برای چاپ استفاده می‌شود که خروجی را طبق دستورات داده شده نمایش دهد. دستور بعد از `printf` برای برگرداندن ترمینال به حالت عادی است.

برای نداشتن مشکل در خروجی نیاز به رفرش ترمینال داریم که برای اینکار از تابع پایین استفاده می‌کنیم.

```
reset_term() {
    printf '\e[2J\e[%dH\e[?7l\e[?25l' "$rows"
}
```

در اینجا `rows` تعداد سطرهای ترمینال است.

برای خواندن یک کلید از دستور زیر استفاده می‌شود.

```
read_keys() {
    read -rsn1
    [[ $REPLY == $'\e' ]] && read -rsn2
    key="${REPLY^^}"
}
```

همانطور که معلوم است از سوییچ `rsn1` برای بی‌نیازی از زدن اینتر استفاده می‌شود. با توجه به اینکه مقدار مورد نظر در متغیری ریخته نمی‌شود از `REPLY` استفاده می‌شود.

```
prompt() {
    printf '\e[H\e[2K\e[?7h\e[?25h%b ' "$1"
}
```

از `prompt` هم برای چاپ یک نوشته استفاده می‌شود. با استفاده از پیام پس از `printf` ترمینال ما پاک شده و پیام بعدی که در آرگومان است چاپ می‌شود.

با استفاده از تابع پایین ما فایل‌های مخفی را در فایل منیجر خود نمایش می‌دهیم.

```
hidden_toggle() {
    [[ $(shopt -p dotglob) =~ -u ]] && shopt -s dotglob|| shopt -u dotglob
    get_files
}
```

تابع `get_files` بعدا توضیح داده می‌شود.

تابع `change_dir` هنگام عوض شدن دایرکتوری‌ای که موقعیت ماست استفاده می‌شود. همانطور که می‌بینید یک متغیر استفاده شده که دایرکتوری اول را به این متغیر می‌ریزد. این دایرکتوری به عنوان نشان‌گذاری شده نمایش داده می‌شود. وضعیت ما هم به `marked` تغییر پیدا می‌کند.

```
change_dir() {
    if [[ ${marked-} == \* ]]
    then
        cd -- -|| return
        unset status mark marked
    else
        cd -- "${1:-$marked}" || return
    fi
    get_files
}
```

برای باز کردن یک فایل از تابع پایین استفاده می‌شود:

```
file_open() {
    bar='[←]back [→]open [↑]exec'
    printf '\e[?1049l\e[2J\e[E\e[?7l'
    mapfile -tn 250 <"$marked"&& printf '%s\n' "${MAPFILE[@]}"
    status='viewing'
    hud
}
```

در اینجا یک تسک بار درست کردیم و وضعیت یا `status` را هم به `viewing` تغییر دادیم. همچنین `hud` را فراخوانی کردیم.

برای بستن فایل‌ها هم از تابع پایین استفاده می‌کنیم. همانطور که دیده می‌شود می‌بینیم که وضعیت دوباره به `marked` تغییر پیدا کرده. همچنین دوباره همانند اول برنامه توابع `init_term` و `draw_files` فراخوانی می‌شود. در نهایت هم `hud` فراخوانی میگردد.

از تابع پایین هم برای اجرای یک فایل استفاده می‌شود. فایل باید اجرایی باشد. این کار با استفاده از ابزار `bash` صورت می‌گیرد. وضعیت به `executed` تغییر می‌کند. قبل از آن هم ترمینال ریست می‌شود.

```
file_exec() {
    reset_term && bash "$marked"
    status='executed'
    hud
}
```

برای ساخت فایل از تابع زیر استفاده می‌شود.

```
make_file() {
    prompt 'New file name:'
    read -re
    if :>"$REPLY"
    then
    mark="9m$REPLY" status='created'
    get_files
    else
    status='error' mark="9m$REPLY exist"
    hud
    fi
}
```

در اینجا از تابع `prompt` که در بالاتر توضیح داده شده استفاده کرده‌ایم. باز هم چون از متغیر در `read` استفاده نکرده ایم از متغیر `REPLY` استفاده کرده‌ایم. وضعیت به `created` تغییر می‌کند. اگر خطایی مشاهده شود وضعیت به `error` تغییر می‌کند و در نهایت خارج می‌شویم.

برای ساخت فایل از `:>rahmat.txt` استفاده می‌کنیم.

```
make_dir() {
    prompt 'New directory name:'
    read -re
    if mkdir "$REPLY"
    then
    status='created' mark="4m$REPLY\e[m/"
    get_files
    else
    status='error' mark="4m$REPLY\e[m/ exist"
    hud
    fi
}
```

برای ساخت پوشه یا دایرکتوری هم از تابع بالا استفاده می‌کنیم. دقیقا همانند ساخت فایل.

برای حذف هم از تابع زیر استفاده می‌کنیم.

```
file_del() {
    prompt "Do you want to delete \e[3$mark\e[m? [y/n]:"
    read -rsn1
```

```
    if [[ ${REPLY,,} == y ]]
    then
    del="$path/$marked"
    rm -fr "${del-}"&& status='deleted'
    [[ $marked == "${PWD##*/}" ]] && change_dir ../|| get_files
    else
    printf '\e[2K'
    fi
}
```

در اینجا اول پیام آیا میخواهید فلان فایل یا دایرکتوری را حذف کنید نمایش داده می‌شود. آدرس فایل یا دایرکتوری در متغیر مارک دخیره می‌شود. پس از نمایش پیغام با استفاده از read میخواهیم یک کلید فشرده شود که اگر آن y بود فایل یا دایرکتوری حذف شود و دایرکتوری ما عوض شده و دوباره لیست فایل‌ها و فولدر ها نمایش داده شود.

از تابع زیر هم برای اجرای کامند ها استفاده می‌شود.

```
comm_exec() {
    prompt ':'
    read -re
    reset_term
    if bash -c "$REPLY"
    then
    status='executed'
    else
    prompt 'Command not found'
    status='error'
    fi
    mark="3m$REPLY" bar='[←]back'; hud
}
```

همانطور که مشخص است اینکار با زدن دکمه : هنگامی که فایل منیجر به دنبال خواندن دکمه است امکان پذیر است. پس از خوانده شدن ترمینال ریست شده و اگر کامند قابل اجرا شدن بود اجرا میگردد.

یکی از توابع مهم کد get_files است که به وضوح استفاده شده است. در این تابع فایل‌ها و فولدرها نمایش داده می‌شوند. این فایل‌ها یا فولدرها با توجه به نوعشان در رنگ‌های متفاوتی نمایش داده می‌شود.

```
get_files() {
    unset files
    IFS=$'\n'
    [[ $PWD == / ]] && PWD=
    for fp in "$PWD"/*
    do
        file="${fp##*/}"
        if [[ -h $fp ]]
        then
            [[ $TERM =~ 256 ]] && color='8;5;42' || color='6;1'
            file+='\e[m@'
        elif [[ -d $fp ]]
        then
            [[ $TERM =~ 256 ]] && color='8;5;147' || color='4;1'
            file+='\e[m/'
        elif [[ -x $fp|| $fp == *'.sh' ]]
        then
            [[ $TERM =~ 256 ]] && color='8;5;210'|| color='2;1'
            file+='\e[m*'
        else
        [[ $TERM =~ 256 ]] && color='8;5;248' || color='7;2'
        fi
        files+=("${color}m$file")
    done
    reverse files
    filesTwo=("${files[@]}") fileCount="${#filesTwo[@]}"
    draw_files && cursor="$rows"
}
```

در این تابع از تابع `reverse` استفاده شده است. در نهایت هم تابع `draw_files` استفاده شده و کرسر را هم برابر تعداد سطرها قرار می‌دهیم.

تابع `draw_files` به صورت زیر است:

```
draw_files() {
    unset hist; i=0 && reset_term
    printf '\e[3%b\e[m\n' "$rows" "${files[@]}"
    bar='[←]back [→]open [q]uit'
    hud
}
```

این تابع در واقع برای نمایش فایل‌ها است. یک جورایی ادامه تابع `get_files` می‌باشد.

تابع کرسر هم در واقع برای نمایش عملیات انتخاب یک فایل یا فولدر است:

```
cursor() {
    (( fileCount > rows )) && {
        if (( ${#files[@]} > rows && cursor < 1 ))
        then
            cursor="$rows"
            files=("${files[@]:0:${#files[@]}-$rows}")
            draw_files
        elif (( cursor > rows ))
        then
            cursor=1
            files=("${filesTwo[@]:0:${#files[@]}+$rows}")
            draw_files
        fi
        (( rows-cursor == ${#files[@]} ))&& cursor="$rows"
    } || {
        ((cursorMin=LINES-fileCount))
        if (( cursor > rows ))
        then
            cursor="$cursorMin"
        elif (( cursor < cursorMin ))
        then
            cursor="$rows"
        fi
    }
    hover="${files[$cursor-$LINES]}"
    printf '\e[%dH\e[4%b\e[m' "$cursor" "$hover"
    (( fileCount == 1 )) || {
        hist+=("${cursor}H\e[3${hover}")
        (( i )) && {
            printf '\e[%b\e[m' "${hist[0]}"
            hist=("${hist[@]:1}")
        } || i=1
    }
}
```

اگر تعداد فایل‌ها از تعداد سطرهای ترمینال بیشتر باشد به صورت پارت پارت نمایش داده می‌شود. وقتی از || استفاده شده به این معنی است که اگر کامند قبل از دوخط اجرا نشود یا خروجی مقداری مثبت باشد (به همان معنی اجرای غلط) آنگاه بعدی اجرا می‌شود وگرنه اجرا نمی‌شود.

در اینجا hover هم به معنای سلکشنی است که با عملیات‌های انتقال به بالا یا پایین منتقل می‌شود.

```
hud() {
    printf '\e[%dH\e[44mRahmat FML\e[m%s\e[3%b\e[m %s' "$LINES" "${status:+
${status^} : }" "${mark:-  }" "$bar"
}
```

تابع hud هم که در تعدادی بالا استفاده شده برای نمایش تسک بار پایین فایل منیجر است.

```
keymap() {
    read_keys
    case $key in
    :)
        comm_exec
        for((;;)) {
        read_keys
        case $key in
            :) comm_exec;;
            H|\[D) draw_files&& break;;
        esac
        }
    ;;
    A) hidden_toggle
    ;;
    D) make_dir
    ;;
    F) make_file
    ;;
    X|\[3) [[ $marked ]] && file_del
    ;;
    H|\[D) change_dir ../
    ;;
    J|\[B) ((cursor++))
    ;;
    K|\[A) ((cursor--))
    ;;
    L|\[C|'')
        status='marked' mark="$hover" path="$PWD"
        marked="${mark#[0-9]*m}" marked="${marked%\\e[m?}"
        change_dir|| {
            file_open
            for((;;)) {
                read_keys
                case $key in
                H|\[D) draw_files && break
                ;;
```

```
                K|\[A) file_exec || return
                ;;
                L|\[C|'') "${VISUAL:-${EDITOR:-vi}}" "$marked"
                ;;
                esac
            }
            file_close
        }
    ;;
    Q) end
    ;;
    esac
}
```

تابع بالا یا keymap هم برای گرفتن یک کلید در صفحه اول یا اصلی فایل منیجر است. اول این تابع از دیگر تابع read_keys استفاده شده. سپس با یک case این کلید دریافت شده با توجه به کلیدی که هست تابع مورد نظر را فرا می‌خواند.

```
trap end 2
trap 'init_term && get_files' 28
```

trap چیه؟

اگر مقدار زیادی کد bash نوشته باشید، احتمالاً با دستور trap برخورد کرده اید. Trap به شما این امکان را می دهد که سیگنال ها را بگیرید و کد را در زمان وقوع آنها اجرا کنید. سیگنال ها اعلان های ناهمزمانی هستند که در صورت وقوع رویدادهای خاص به اسکریپت شما ارسال می شوند. بیشتر این اعلان‌ها برای رویدادهایی هستند که امیدوارید هرگز رخ ندهند، مانند دسترسی نامعتبر به حافظه یا تماس سیستمی بد. با این حال، یک یا دو رویداد وجود دارد که ممکن است منطقی بخواهید با آنها مقابله کنید. همچنین رویدادهای "کاربر" در دسترس هستند که هرگز توسط سیستم تولید نمی شوند و می توانید برای سیگنال دادن به اسکریپت خود ایجاد کنید. Bash همچنین یک سیگنال شبه به نام "EXIT" ارائه می دهد که هنگام خروج از اسکریپت شما اجرا می شود. این می تواند برای اطمینان از اینکه اسکریپت شما مقداری پاکسازی در هنگام خروج اجرا می کند استفاده شود.

کد پایین هم برای اجرای دستورات یا توابع اصلی است:

```
init_term && get_files
for((;;)) {
    cursor
    keymap
}
```

در اینجا اول توابع `init_term` و سپس `get_files` فراخوانی شده برای مهیا کردن شرایط اجرای بقیه کدها.

سپس در یک حلقه بینهایت توابع `curser` و `keymap` فراخوانی می‌شود.

```
reverse() {
    local -n foo="$1"
    shopt -s extdebug
    bar()( printf '%s\n' "${BASH_ARGV[@]}" )
    foo=($(bar "${foo[@]}")) && unset "foo[-1]"
    shopt -u extdebug
}
```

تابع بالا که در تابع `get_files` به صورت زیر استفاده شده است:

Reverse files

برای نمایش فایل‌ها استفاده می‌شود. در اینجا مقادیر `files` را در آرایه `foo` قرار میدهیم که به معنای `footer` است. در `${foo[@]}` هم کل مقدار آرایه چاپ می‌شود.

Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.

بیشتر برنامه نویسان خوب برنامه نویسی می کنند نه به این دلیل که انتظار دارند دستمزد دریافت کنند

یا از طرف مردم تحسین شوند، بلکه به این دلیل که برنامه نویسی سرگرم کننده است •

Linus Torvalds

creator of the Linux OS