리눅스 기본 명령어

date	날짜출력
hostname	서버이름 출력
uname	운영체제 출력
uname -a	운영체제 상세정보 출력
whoami	유저이름 출력
ls	디렉터리 내용을 리스트
ls -a	모든 파일과 디렉터리 리스트
ls -asl	모든파일 자세히 리스트
passwd	패스워드 변경
clear	화면 청소
man (명령어)	명령어 매뉴얼
pwd	현재 작업 디렉터리를 프린트
mkdir (폴더)	새디렉터리 생성
cd (경로)	현재 작업 디렉터리를 이동
cat	키보드 입력 내용을 출력
cat (파일)	파일 내용 출력
cat > (파일)	키보드 입력 내용을 파일에 저장
more (파일)	파일을 페이지 단위로 화면에 출력
head (파일)	파일의 앞부분(10줄)을 출력
tail (파일)	파일의 뒷부분(10줄)을 출력
wc (파일)	파일의 줄, 단어, 문자 갯수 출력
cp (파일1) (파일2)	파일1을 파일2에 복사
cp (파일) (폴더)	파일의 복사본을 폴더 내에 만듬
cp -i	대화형 옵션
mv (파일1) (파일2)	파일1의 이름을 파일2로 변경
mv (파일) (폴더)	파일을 지정된 폴더로 이동
rm (파일)	파일 삭제
rm -r (폴더)	폴더 삭제
ln [-s] (파일1) (파일2)	이름:파일2 파일1 바로가기 생성 -s 옵션은 심볼릭 링크 (바로가기)
chmod (oct 3)	접근권한 변경
chown (user) (파일)	파일이나 폴더의 소유자 변경
chown (user) (폴더)	
chgrp (group) (파일)	파일이나 폴더의 그룹 변경
chgrp (group) (폴더)	

고정 IP 설정

가상머신 기본 Network 설정은 DHCP
*DHCP : IP 자동할당 재부팅시 ip 변경 가능

고정 IP 설정 방법
/etc/netplan/00-installer-config.yaml 파일 수정

```
addresses:
        - 192.168.~.~/24
~~~~~~~~
routes:
        - to: default
        via: 192.168.0.1
```

yaml 파일이 존재하지 않을 경우
sudo netplan generate

파일 수정 후
sudo netplan apply

리눅스 파일 종류

* 일반파일
* 디렉터리(폴더)
* 특수 파일
  * 물리적인 장치들을 파일로 표현
  * 리눅스 운영체제 안에서 사용되는 프로세스간 통신 수단 등을 파일로 표현
* 심볼릭 링크 파일
  * 어떤파일을 가리키는 또 하나의 경로명을 저장하는 파일

접근권한 표현 : 8진수

* 접근권한 8진수 변환

사용자	그룹	기타
rwx	r-x	r-x
111	101	101
7	5	5

ex) chmod 644 (파일)
= user(rw-) group(r—) other(r—)

접근권한 표현 : 기호

구분	기호와 의미
사용자 범위	u(소유자), g(그룹), o(기타), a(all)
연산자	+(추가), -(제거), =(설정)
권한	r,w,x

ex) chmod u+r
= user(r—)

출력 재지정

* 명령어의 표준 출력 내용을 모니터 대신에 파일에 저장

명령어 > 파일
ex) who > names.txt

출력 추가

* 명령어의 표준 출력 내용을 모니터 대신에 파일에 추가

명령어 >> 파일
ex) who >> names.txt

입력 재지정

* 명령어의 표준입력을 키보드 대신에 파일에서 받음

명령어 < 파일
ex) wc < list1.txt

문서 내 입력

* 명령어의 표준 입력을 키보드 대신에 단어와 단어 사이의 입력 내용으로 받음
* 보통 스크립트 내에서 입력을 줄 때 사용

명령어 << 단어
ex) wc << end

파이프

* | : 두개 이상의 명령어의 입출력을 연결

명령어1 | 명령어2
ex) ls | sort -r
(명령어1 한거에 명령어 2)

시스템 호출과 C 라이브러리 함수

* 시스템 호출(System Calls)
  * Unix 커널에 서비스 요청하는 호출
  * C 함수처럼 호출 가능

* C 라이브러리 함수(Library Functions)
  * C 라이브러리 함수는 보통 시스템 호출을 포장해 놓은 함수
  * 보통 라이브러리 함수 내부에서 “시스템 호출”을 호출함

시스템 호출

* 리눅스/유닉스 커널에 서비스 요청을 위한 프로그래밍 인터페이스
* 응용 프로그램은 시스템 호출을 통해 커널 서비스 요청가능

파일

* C 프로그램 파일
  * 변수에 저장된 데이터나 프로그램을 통해 생성한 데이터를 영속적으로 저장하기 위한 목적
* 유닉스 파일
  * 모든 데이터를 연속된 바이트로 저장

C언어의 파일 종류

* 텍스트 파일
  * 파일에 저장되는 바이트의 연속이 ASCII code나 Unicode등의 문자 형식
  * “한 줄의 끝”을 나타내는 코드는 파일이 읽어 들여질 때 C 내부 형식으로 변환 되며, 파일에 저장할때는 유닉스 파일 형식으로 변환
  * \r\n ~~ \n
* 이진 파일
  * 모든 데이터는 있는 그대로 바이트의 연속으로 저장
  * 메모리에 저장된 변수 값을 그대로 파일로 저장 가능

파일 입출력 : fread(), fwrite()

* C 언어의 파일 입출력 과정
  * 파일 열기: fopen() 함수 사용
  * 파일 입출력 : 다양한 파일 입출력 함수 (fread, fwrite 등) 사용
  * 파일 닫기 : fclose()

파일 열기 : fopen()

* 파일을 접근하고 읽기/쓰기 하기 위해서는 반드시 파일 열기(fopen)를 먼저 수행해야 함
* 파일 열기에 성공하면 FILE 구조체에 대한 포인터 반환됨
  * FILE 구조체 포인터 열린 파일을 의미
* ex)

```
FILE *fopen(const char *filename, const char *mode);
// - const char *filename: 파일명에 대한 포인터
// - const char *mode: 모드로 파일을 여는 형식

Ex 1)
FILE *fp;
fp = fopen(“~/sp/text.txt”, “r”);
if (fp == NULL) {
   printf("파일 열기 오류\n”);
}
Ex 2)
fp = fopen("outdata.txt", “w”);
fp = fopen("outdata.txt", “a”);
```

fopen() : 텍스트 파일 열기

* const char *mode

모드	의미	파일이 없으면	파일이 있으면
“r”	읽기 전용 (read)	NULL 반환	정상 동작
“w"	쓰기 전용 (write)	새로 생성	기존 내용 삭제
“a”	추가 쓰기 (append)	새로 생성	기존 내용 뒤에 추가
“r+”	읽기와 쓰기	NULL 반환	정상 동작
“w+”	읽기와 쓰기	새로 생성	기존 내용 삭제
“a+”	추가를 위한 읽기와 쓰기	새로 생성	기존 내용 뒤에 추가

FILE 구조체

* 파일 관련 시스템 호출
  * 파일 디스크립터로 열린 파일을 지칭
* C 표준 입출력 함수
  * fopen() 함수로 파일을 열면 FILE 구조체 포인터 (FILE*)가 반환됨
  * 열린 파일에 대한 정보를 저장하는 FILE 구조체에 대한 포인터
  * FILE 포인터를 표준 입출력 함수들의 인수로 전달해야 함
  * #include <stdio.h>
* FILE구조체
  * 하나의 스트림에 대한 정보를 포함하는 구조체
  * 버퍼에 대한 포인터, 버퍼 크기…
  * 파일 디스크립터
* 열린 파일의 현재 상태를 나타내는 필드 변수
* 특히, 파일 입출력에 사용되는 버퍼 관련 변수
* ex)

```
typedef struct {
   int cnt;                  // 버퍼의 남은 문자 수
   unsigned char*base;       // 버퍼 시작
   unsigned char*ptr;        // 버퍼의 현재 포인터
   unsigned flag;            // 파일 입출력 모드
                             // (_IOFBF, _IOLBF, _IONBUF, _IOEOF,
                             // _IOERR, _IOREAD, _IOWRT)
   int fd;                   // 열린 파일 디스크립터
} FILE;                      //FILE 구조체
```

표준 입력/출력/오류

* 표준 I/O 스트림
  * 프로그램이 시작되면 자동으로 open되는 스트림
    * stdin, stdout, stderr
  * FILE 구조체 포인터

표준 입출력 포인터	설명	가리키는 장치
stdin	표준 입력에 대한 FILE 포인터	키보드
stdout	표준 출력에 대한 FILE 포인터	모니터
stderr	표준 오류에 대한 FILE 포인터	모니터

파일 닫기 : fclose()

* 파일을 열어서 사용한 후에는 파일을 닫아야 함
* ex)

```
int fclose(FILE *fp);
// - fp : dopen 함수에서 반환 받은 포인터
// - 파일 닫기에 성공하면 0, 오류일 때는 EOF(-1) 반환

EX)
fclose(fp);
```

텍스트 파일

파일 입출력 함수

표준 입출력 함수	파일 입출력 함수	기능
getchar()	fgetc(), getc()	문자단위로 입력하는 함수
putchar()	fputc(), putc()	문자단위로 출력하는 함수
gets()	fgets()	문자열을 입력하는 함수
puts()	fputs()	문자열을 출력하는 함수
scanf()	fscanf()	자료형에 따라 자료를 입력하는 함수
printf()	fprintf()	자료형에 따라 자료를 출력하는 함수

문자 단위 입출력 : fgetc(), fputc()

* 파일에 문자 단위 입출력 수행
  * int fgetc(FILE *fp);
    * fp가 지정한 파일에서 한 문자를 읽어서 반환 파일 끝에 도달했을 경우에는 EOF(-1) 반환
* int fputc(int c, FILE *fp);
  * fp가 가리키는 파일에 한 문자씩 출력
  * 리턴값으로 출력하는 문자 리턴
  * 출력시 오류가 발생하면 EOF(-1) 반환

cat.c

```
#include <stdio.h>
/* 텍스트 파일 내용을 표준출력에 출력 */
int main(int argc, char *argv[])
{
   FILE *fp;
   int c;
   if (argc < 2)
      fp = stdin; // argv가 없으면 표준입력 사용
   else fp = fopen(argv[1],"r"); // 읽기 전용으로 파일 열기

   c = getc(fp); // 파일로부터 문자 읽기
   while (c != EOF) { // 파일끝이 아니면
      putc(c, stdout); // 읽은 문자를 표준출력에 출력
      c = getc(fp); //파일로부터 문자 읽기
   }
   fclose(fp);
   return 0;
}
```

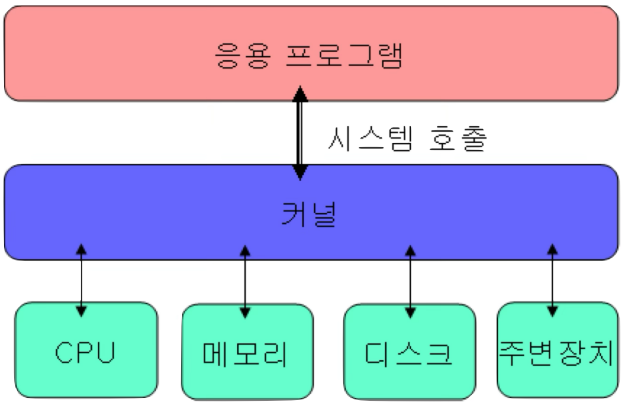
copy.c

파일입출력

시스템 호출

컴퓨터 시스템 구조

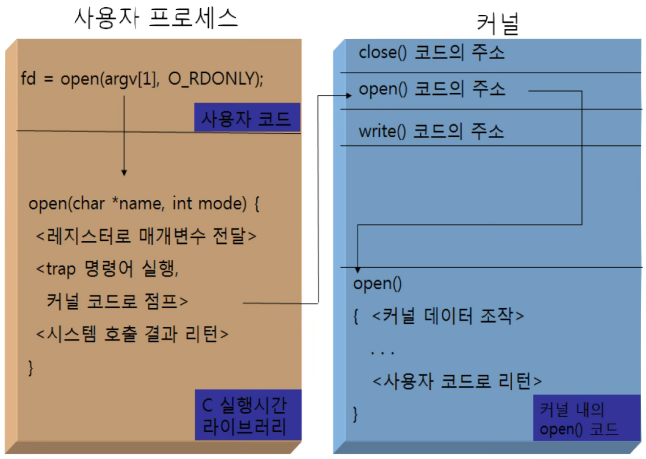
* 유닉스/리눅스 커널
  * 파일관리
  * 프로세스 관리
  * 메모리 관리
  * 통신 관리
  * 주변장치 관리



시스템 호출

* 개념
  * 커널에 서비스 요청을 위한 프로그래밍 인터페이스
  * 응용 프로그램은 시스템 호출을 통해서 커널에 서비스 요청 (운영체제가 제공하는 기능 사용)

시스템 호출 과정



시스템 호출 요약

주요 자원	시스템 호출
파일	open(), close(), read(), write(), dup(), lseek() 등
프로세스	fork(), exec(), exit(), wait(), getpid(), getppid() 등
메모리	malloc(), calloc(), free() 등
시그널	signal(), alarm(), kill(), sleep() 등
프로세스 간 통신	pipe(), socket() 등

파일

* 커널의 파일
  * 연속된 바이트의 나열
  * 특별한 다른 포맷을 정하지 않음
  * 디스크 파일뿐만 아니라 외부 장치에 대한 인터페이스
  * C 라이브러리 함수에서 처럼 텍스트/바이너리 구분 없음

파일열기 : open()

* open() 시스템 호출
* ex)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntil.h>
int open (const char *path, int flag, [ mode_t mode ]);
* 파일 열기에 성공하면 파일 디스크립터를 , 실패하면 -1을 반환
```

  * 파일을 사용하기 위해서는 먼저 open() 시스템 호출을 이용하여 파일을 열어야 함
  * 파일 디스크립터는 열린 파일을 나타내는 번호
* oflag
  * O_RDONLY : 읽기 모드, read() 호출은 사용 가능
  * O_WRONLY : 쓰기 모드, write() 호출은 사용 가능
  * O_RDWR : 읽기/쓰기 모드, read(), write() 호출 사용 가능
  * O_APPEND : 데이터를 쓰면 파일 끝에 추가
  * O_CREATE : 해당 파일이 없는 경우에 생성, mode는 생성 파일 사용 권한
  * O_TRUNC : 파일이 이미 있는 경우 내용 삭제
  * O_EXCL : O_CREATE와 함께 사용되며 해당 파일이 이미 있으면 오류
  * O_NONBLOCK : 넌-블로킹 모드로 입력/출력
  * O_SYNC : write() 시스템 호출시 디스크에 물리적으로 쓴 후 종료 (버퍼사용 X)
  * ex)

```
fd = open(“account”, O_RDONLY);
fd = open(argv[1],  O_RDWR);
fd = open(argv[1], O_RDWR | O_CREATE, 0600);
fd = open(“tmpfile”, O_WRONLY | O_CREATE | O_TRUNC, 0600);
fd = open(“/sys/log”, O_WRONLY|O_APPEND|O_CREATE, 0600);
if((fd = open(“tmpfile”, O_WRONLY|O_CREATE|O_EXCL, 0666))==-1)
```

* fopen.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char *argv[]){
   int fd;
   if ((fd = open(argv[1],O_RDWR))==-1)
      printf(“파일 열기 오류\n”);
   else printf(“파일 %s 열기 성공 : %d\n”, argv[1], fd);
   close(fd);
   exit(0);
   return 0;
}
```

파일 생성 : create()

* create() 시스템 호출

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int create ( const char *path, node_t mode );
* 파일 생성에 성공하면 파일 디스크립터를, 실패하면 -1을 반환
```

  * path가 나타내는 파일을 생성하고 쓰기 전용 오픈
  * 생성된 파일의 사용권한은 mode로 정함
  * 기존 파일이 있는 경우에는 그 내용을 삭제
  * 다음 시스템 호출과 동일

```
open(path, WRONLY | O_CREATE | O_TRUNC, mode);
```

파일 닫기 : close()

* close() 시스템 호출

```
#include <unistd.h>
int close( int fd );
* fd가 나타내는 파일을 닫는다.
* 성공하면 0, 실패하면 -1을 반환한다.
```

데이터 읽기 : read()

* read() 시스템 호출

```
#include <unistd.h>
ssize_t read ( int fd, void *buf, size_t nbytes );
* 파일 읽기에 성공하면 읽은 바이트 수, 파일 끝을 만나면 0, 실패하면 -1을 반환한다,
```

  * fd가 나타내는 파일에서 nbytes 만큼의 데이터를 읽어서 buf에 저장
* 파일 크기 계산 : fsize.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define BUFSIZE 1024
/*파일크기를계산한다*/
int main(int argc, char *argv[]) {
   char buffer[BUFSIZE];
   int fd;
   ssize_t nread;
   long total = 0;
   if ((fd = open(argv[1], O_RDONLY)) == -1) perror(argv[1]);
   /*파일의끝에도달할때까지반복해서읽으면서파일크기계산*/
   while( (nread = read(fd, buffer, BUFSIZE)) > 0)
      total += nread;
   close(fd);
   printf ("%s 파일 크기 : %ld 바이트 \n", argv[1], total);
   exit(0);
}
```

데이터 쓰기 : write()

* write() 시스템 호출

```
#include <unistd.h>
ssize_t write (int fd, void *buf, size_t nbytes);
* 파일에 쓰기를 성공하면 실제 쓰여진 바이트 수 반환, 실패하면 -1을 반환
```

* buf에 있는 nbytes 만큼의 데이터를 fd가 나타내는 파일에 쓰기

* copy.c

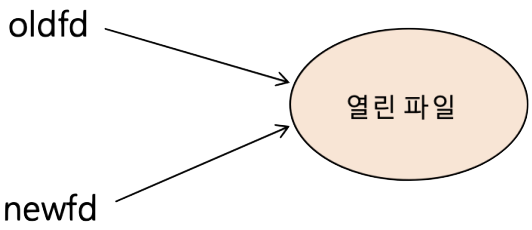
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
/*파일복사프로그램*/
int main(int argc, char *argv[]) {
   int fd1, fd2, n;
   char buf[BUFSIZ];
   if (argc != 3) {
      fprintf(stderr,"사용법: %s file1 file2\n", argv[0]);
      exit(1);
   }
   if ((fd1 = open(argv[1], O_RDONLY)) == -1) {
      perror(argv[1]);
      exit(2);
   }
   if ((fd2 =open(argv[2], O_WRONLY | O_CREAT|O_TRUNC, 0644))
== -1) {
      perror(argv[2]);
      exit(3);
   }
   while ((n = read(fd1, buf, BUFSIZ)) > 0)
   write(fd2, buf, n); // 읽은 내용을 쓴다.
   exit(0);
   }
```

파일 디스크립터 복제

* dup()/dup2() 호출

```
#include <unistd.h>
int dup(int oldfd);
* oldfd에 대한 복제본인 새로운 파일 디스크립터를 생성하여 반환한다.
* 실패하면 -1을 반환한다.
int dup2(int oldfd, int newfd);
* oldfd을 newfd에 복제하고 복제된 새로운 파일 디스크립터를 반환한다.
* 실패하면 -1을 반환한다.
```

* 기존의 파일 디스크립터를 복제
* oldfd와 복제된 새로운 디스크립터는 하나의 파일 공유



* dup.c

```
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
int main(){
   int fd1, fd2;
   if((fd1 = creat("myfile", 0600)) == -1)
   perror("myfile");
   write(fd1, "Hello! Linux", 12);
   fd2 = dup(fd1);
   write(fd2, "Bye! Linux", 10);
   exit(0);
}
```

임의 접근 파일

임의 접근과 파일 위치 포인터

* 파일 위치 포인터
  * 파일 내에 읽거나 쓸 위치인 현재 파일 위치

파일 위치 포인터

* 임의 접근 파일(random access file)
  * 파일 내의 원하는 지점으로 바로 이동하여 읽기/쓰기 수행

임의 접근 : lseek()

* lseek() 시스템 호출

```
#include <stdio.h>
off_t seek (int fd, off_t offset, int whence );
```

* 이동에 성공하면 현재 위치를 반환하고 실패하면 -1을 반환한다.
* 임의의 위치로 파일 위치 포인터를 이동시킬 수 있다.

파일 위치 포인터 이동 EX

* 파일 위치 이동
  * lseek(fd, 0L, SEEK_SET); 파일 시작으로 이동
  * lseek(fd, 100L, SEEK_SET); 파일 시작에서 100바이트 위치로
  * lseek(fd, 0L, SEEK_END; 파일 끝으로 이동
* 레코드 단위로 이동
  * lseek(fd, n*sizeof(record), SEEK_SET); n+1 번째 레코드 시작위치로
  * lseek(fd, sizeof(record), SEEK_CUR); 다음 레코드 시작위치로
  * lseek(fd, -sizeof(record), SEEK_CUR); 전 레코드 시작위치로
* 파일끝 이후로 이동
  * lseek(fd, sizeof(record), SEEK_END); 파일끝에서 한 레코드 다음 위치로

레코드 저장 EX

* write(fd, &record1, sizeof(record));
* write(fd, &record2, sizeof(record));
* lseek(fd, sizeof(record), SEEK_END);
* write(fd &record3, sizeof(record));

학생 레코드 파일 EX

* 임의 접근을 이용한 학생 레코드 파일
  * 저장(dbcrate.c)
  * 질문(dbquery.c)
  * 수정(dbupdate.c)
* 학생 레코드 저장 위치
  * 학번을 기준으로 학생 레코드를 해당 위치에 저장
  * 학번(rec.id)에 해당하는 학생 레코드의 위치

$$(rec.id - START\_ID) * sizeof(rec)$$

1001001	1001002		1001004

임의 접근을 이용한 학생 레코드 저장

* dbcreate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "student.h"
/* 학생 정보를 입력받아 데이터베이스 파일에 저장한다. */
int main(int argc, char *argv[])
{
   int fd;
   struct student record;
   if (argc < 2) {
      fprintf(stderr, "사용법 : %s file\n", argv[0]);
      exit(1);
   }
   if ((fd = open(argv[1], O_WRONLY|O_CREAT|O_EXCL, 0640)) == -1)
{
      perror(argv[1]);
      exit(2);
   }
   printf("%-9s %-8s %-4s\n", "학번", "이름", "점수");
    while (scanf("%d %s %d", &record.id, record.name, &record.score)
== 3) {
      lseek(fd, (record.id - START_ID) * sizeof(record), SEEK_SET);
      write(fd, (char *) &record, sizeof(record) );
   }
   close(fd);
   exit(0);
}
```

* student.h

```
#define MAX 24
#define START_ ID 1001001

struct student {
   chr name[MAX];
   int id;
   int score;
};
```

* dbquery.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "student.h"
/* 학번을 입력받아 해당 학생의 레코드를 파일에서 읽어 출력한다. */
int main(int argc, char *argv[])
{
   int fd, id;
   char c;
   struct student record;
   if (argc < 2) {
      fprintf(stderr, "사용법 : %s file\n", argv[0]);
      exit(1);
   }
   if ((fd = open(argv[1], O_RDONLY)) == -1){
      perror(argv[1]);
      exit(2);
   }
   do{
      printf("\n검색할 학생의 학번 입력:");
      if (scanf("%d", &id) == 1) {
         lseek(fd, (id-START_ID)*sizeof(record), SEEK_SET);
         if ((read(fd, (char *) &record, sizeof(record)) > 0) &&
(record.id != 0))
            printf("이름:%s\t 학번:%d\t 점수:%d\n", record.name,
record.id, record.score);
         else
            printf("레코드 %d 없음\n", id);
      }
      else
         printf("입력오류");
      printf("계속하겠습니까?(Y/N)");
      scanf("%c", &c);
   } while (c =='Y');

   close(fd);
   exit(0);
}
```

쉘 프로그래밍 문법

변수

* 쉘 변수의 특징
  * 변수를 사용하기 전에 선언하지 않음
  * 변수의 값은 항상 문자열로 간주됨
  * 대소문자를 구별함
  * $를 변수 앞에 붙여서 변수의 값을 접근
  * read 명령어를 이용하여 사용자입력을 변수에 저장 가능
* 변수와 따옴표
  * 변수의 값이 하나이상의 공백문자를 포함하면 따옴표를 이용해 표시해야 함
  * “ ”와 ‘ ’ 모두 사용가능
    * “ “ : 변수값에 $변수명이 포함될때 변수를 저장된 값으로 대체
    * ‘ ‘ : $변수명을 그대로 출력
    * \ : $의 의미를 제거
* 환경변수
  * 쉘 스크립트가 시작할 때 환경으로부터 얻은 값으로 초기화 되는 변수
  * 일반적으로 대문자 사용
  * 계정 사용자에 의해서 추가 가능
  * 기본적인 환경 변수
    * $HOME : 현재 사용자의 홈 디렉토리
    * $PATH : 명령을 검색할 디렉토리들을 콜론으로 구분해 놓은 목록
    * $PS1 : 명령 프롬프트. 추가적인 입력을 받아들이는 프롬프트(일반적으로 >)
    * $PS2 : 두번째 프롬프트. 추가적인 입력을 받아들이는 프롬프트 (일반적으로 >)
    * $IFS : 입력 필드 구분자. 쉘이 입력을 읽을때 단어들을 구분하는 문자 목록
    * $0 : 쉘 스크립트의 이름
    * $# : 전달된 매개변수의 개수
    * $$: 쉘 스크립트의 프로세스 ID. 주로 임시 파
    * 일 이름을 유일하게 생성 하기 위해 사용됨
  * 매개변수
    * 스크립트를 실행할 때 전달되는 변수
    * 매개변수가 전달되지 않더라도 $#은 0
    * 매개변수 종류
      * $1, $2, … : 스크립트에 주어진 매개변수
      * $* : 모든 배개변수들의 목록을 하나의 변수로 나타냄. 환경 변수 IFS에 저장된 첫번째 문자로 매개변수 구분
      * $@ : $*의 확장형. IFS가 비어 있다면 매개 변수들은 함께 나열됨 IFS가 설정되어도 구분하여 나타냄

조건

* test, [ ]
  * ex) test string1 = string2
  * ex) [string1 = string2] ([띄어쓰기 필수])
* 문자열 비교
  * [string] : 빈 문자열이 아니라면 참
  * [string1 = string2] : 두 문자열이 같다면 참
  * [string1 != string2] : 두 문자열이 다르면 참
  * [-n string] : 문자열이 null이 아니라면 참
  * [-z string] : 문자열이 null이라면 참
* 산술비교
  * [a -eq b] : 두 표현식 값이 같다면 참 (EQual)
  * [a -ne b] : 두 표현식 값이 다르면 참 (Not Equal)
  * [a -gt b] : a > b 이면 참 (Greater Then)
  * [a -ge b] : a >= b 이면 참 (Greater Equal)
  * [a -lt b] : a < b 이면 참 (Less Then)
  * [a -le b] : a <= b 이면 참 (Less Equal)
  * [! a] : not a
  * [a -a b] : a and b (And)
  * [a -o b] : a or b (Or)
* 파일조건
  * [-b file] : file이 블럭 디바이스면 참
  * [-c file] : file이 문자 디바이스면 참
  * [-d file] : file이 디렉토리면 참
  * [-e file] : file이 존재하면 참
  * [-f file] : file이 존재하고 정규파일이면 참
  * [-g file] : file이 set-group-id 파일이면 참
  * [-h file] : file이 심볼릭 링크면 참
  * [-L file] : file이 심볼릭 링크면 참
  * [-k file] : file이 Sticky bit가 셋팅되어 있으면 참
  * [-p file] : file이 파이프로 지정되어있으면 참
  * [-r file] : 현재 사용자가 읽을 수 있는 file이면 참
  * [-s file] : file이 비어있지 않으면 참
  * [-S file] : 소켓디바이스이면 참
  * [-t FD] : FD가 열려진 터미널이면 참
  * [-u file] : file 이 set-user-id 파일이면 참
  * [-w file] : 현재사용자가 쓸수있는 파일이면 참
  * [-x file] : 현재사용자가 실행할 수 있는 파일이면 참
  * [-O file] : file의 소유자가 현재사용자이면 참
  * [-G file] : file의 그룹이 현재사용자의 그룹과 같으면 참
  * [file1 -nt F] : file1이 file2보다 새로운 파일이면 참
  * [file1 -ot F] : file1이 file2보다 오래된 파일이면 참
  * [file1 -ef F] : file1이 file2의 하드링크파일이면 참

if 문

* 조건에 따라 수행할 부분을 결정

구조)

```
if 조건
then
   if 조건
   then
      명령어
   else
      명령어
   fi
elif 조건
then
   명령어
else
   명령어
fi
```

ex)

```
if [ $timeofday = “yes” ]
then
   echo “Good morning”
elif [$timeofday = “no” ]
then
   echo “Good afternoon”
else
   echo “Sorry, $timeofday not recognized.”
   exit 1
fi
```

case 문

* 값에 따라 수행될 부분을 결정

구조)

```
case 변수 in
패턴1)
   명령어
   …
   ;;
패턴2)
   명령어
   …
   ;;
*)
   명령어
   …
   ;;
esac
```

ex)

```
case “$timeofday” in
   “yes”) echo “Good morning”;;
   “no”) echo “Good afternoon”;;
   *) echo “Sorry, answer not recognized”;;
esac
```

for 문

* 패턴 리스트 수에 따라 반복을 수행

구조)

```
for 변수 in 값(리스트);
do
   명령어
   …
done
```

ex)

```
for file in $(ls f*);
do
   echo $file
done
```

*(())을 이용하여 C, Java같은 형식의 for문 사용가능

ex)

```
for ((i=0;i<=10;i++));
do
   echo $i
done
```

*{..}을 이용하여 range함수 구현 가능

ex)

```
for i in {1..10};
do
   echo $i
done
```

whlie 문

* 조건이 맞는 동안 반복을 수행

구조)

```
while 조건
do
   명령어
   …
done
```

ex)

```
while [ “$trythis” != “secret” ];
do
   echo “Sorry, try again”
   read trythis
done
```

until 문

* 조건이 맞을때까지 반복을 수행

구조)

```
until 조건
do
   명령어
   …
done
```

ex)

```
until [ ! $try ];
do
   read try >> try.txt
done
```

리스트

* 명령어를 일렬로 연결
* AND 리스트
  * 명령어1 & 명령어2 & 명령어3 & …
  * 왼쪽부터 명령어가 false를 반환할때까지 실행
  * false 반환 받으면 뒤에는 실행안됨
* OR 리스트
  * 명령어1 | 명령어2 | 명령어3 | …
  * 왼쪽부터 명령어가 true를 반환할때까지 실행
  * true 반환 받으면 뒤에는 실행안됨

ex)
```
case “$timeofday” in
  yes | y | Yes | YES) echo “Good morning”;;
  n* | N*) echo “Good afternoon”;;
  *) echo “Sorry, answer not recognized”;;
esacs
```

함수

구조)
* 함수 정의

```
함수이름 (){
  명령어
  …
}
```

* 함수 호출

```
함수이름
```

ex)
```
foo(){
  echo “Function foo is executing”
}

foo
```

* 매개변수 전달
  * 함수이름 매개변수1 …
  * 함수내부에서는 매개변수를 $*, $1, $2 … 로 접근
* 값 반환
  * return 값

ex)
```
foo(){
  echo “Hi, $1”
}

foo james
```

명령수행

* 쉘 스크립트에서는 두가지 형식의 명령 수행 가능
  * 외부명령 : 명령 프롬프트에서 수행할 수 있는 명령
  * 내부명령 : 쉘 내부적으로 구현되어 명령 프롬프트에서는 실행되지 않는 명령
* 명령어들

break	반복문 종료
:	null 명령, true의 별칭
continue	반복 계속
eval	인자의 연산
exec	현재 쉘을 다른 프로그램으로 대체
exit n	종료 코드 반환
	0 성공
	1 ~ 125 에러코드
	127 명령 찾을 수 없음
	128이상 신호발생
export	특정 변수를 하위 쉘의 환경변수로
expo	표현식 연산
printf	echo 상위호환
return	함수 값 반환
set	쉘에 매개변수를 설정
shift	매개 변수의 순서를 이동
trap	신호를 받았을때 작동 지정
find	파일 검색
grep	파일에서 문자열 검색

export

* export 뒤에 명령어를 다음 bash 환경변수로 보낸다.

ex)
* export2

```
echo “$foo”
echo “$bar”
```

* export1

```
foo=“abcd”
export bar=“efgh”
./export2
```

* 실행결과

```
efgh
```

expr, printf

* expr
  * 표현식 수행
    * x=`expr $x + 1`
    * x=$(expr $x + 1)
    * x=$(($x+1))
  * expr로 수행할수 있는 표현식
    * |, &, =, >, >=, <, <=, !=, +, -, *, /, %
* printf
  * printf “형식 문자열” 매개변수1 매개변수2 …
  * 이스케이프 시퀀스
    * \\, \a, \b, \f, \n, \r, \t, \v
  * 변환지정자
    * d, c, s, %
  * ex)
    * printf “%s\n” hello
    * - “hello”
    * printf “%s %d\t%s” Hi There” 15 people
    * - “Hi There 15 people”
    * printf “%05d” 1
    * - 00001

find

* 컴퓨터에서 파일 검색
  * find 경로 옵션 테스트 작동
  * ex) find ./ -name wish -print
* 주요 옵션
  * -depth : 디렉토리 자체를 살펴보기전에 디렉토리의 내용을 검색
  * -follow : 심볼릭 링크를 따라가서 검색
  * -maxdepths -N : 최대 N 수준의 디렉토리를 검색
  * -mount (or -xdev) : 다른 파일 시스템의 디렉토리는 검색하지 않음
* 주요 테스트
  * -atime N : 파일이 N일 이전에 마지막으로 액세스 된것을 검색
  * -mtime N : N일 이전에 수정된 것 검색
  * -name "패턴" : 경로를 제외한 파일의 이름이 주어진 패턴에 일치하는 것 검색
  * -newer otherfile : otherfile 보다 최신인 파일 검색
  * -type C : 파일 형식이 C 인것 (f : 파일, d : 디렉토리)
  * -user 사용자 이름 : 주어진 사용자가 소유한 파일 검색
* 연산자
  * !, -not
  * -a, -and
  * -o, -or
* ex)
  * find . -newer first -print
  * find . -newer first -type f -print
  * find . \(-name "_*" -or -newer first \) -type f -print
* 작동
  * -exec : 특정 명령 실행
  * -ok : exec와 동일, 사용자에게 명령 실행 여부 물어봄
  * -print : 화면에 출력
  * -ls : 현재 파일에 대해서 ls a명령 실행

grep

* 파일에서 문자열 검색
  * grep 옵션 패턴 파일
* 옵션
  * -c : 일치하는 줄을 모두 출력하지 않고, 일치하는 줄의 개수를 출력
  * -E : 확장 표현식 적용
  * -h : 각 출력 줄에 파일이름을 붙이는 작업을 수행하지 않음
  * -i : 대소문자 구별 안함
  * -l : 일치된 줄에 해당하는 파일 명을 나열. 일치된 줄은 출력하지 않음
  * -v : 일치되지 않은 줄만을 선택
* ex)
  * grep print second
  * grep -c print first second
  * grep -c -v print first second

명령 실행

* 쉘 스크립트 안에서 명령을 실행하고 그 결과를 쉘 스크립트에서 사용
  * $(명령)
  * `명령`

산술 확장, 변수 확장

* 변수확장
  * 변수의 일부를 잘라내거나, 변수에 문자를 붙일 수 있음
  * 확장 방법
    * ${변수:-default} : 변수가 널이라면 default 값으로 대체
    * ${#변수} : 변수의 길이
    * ${변수%패턴} : 변수 값 끝에서부터 패턴에 일치하는 부분을 탐색하고 짧은 부분 제거
    * ${변수%%패턴} : 변수 값 끝에서부터 패턴에 일치하는 부분을 탐색하고 긴부분 제거
    * ${변수#패턴} : 변수 값 앞에서부터 패턴에 일치하는 부분을 탐색하고 짧은부분 제거
    * ${변수##패턴} : 변수 값 앞에서부터 패턴에 일치하는 부분을 탐색하고 긴 부분 제거
    * ex)
      abc=document/study/makeFile.sh
      a=""
      a:-default : default
      #abc : 26
      abc%/* : document/study (/makeFile.sh 제거)
      abc%%/* : document (/study/makeFile.sh 제거)
      abc#*/ : study/makeFile.sh (document/ 제거)
      abc##*/ : makeFile.sh (document/study/ 제거)

디버깅

* 일반적인 디버깅 방법
  * 쉘 스크립트의 에러가 발생하면 에러를 포함하고 있는 줄의 번호 출력
  * 에러 원인으 알수 없을 때는 echo를 이용해서 변수의 내용을 출력 하거나 대화모드 쉘 실행 방법을 이용해서 의심나는 부분을 실행 해봄
* sh의 실행 옵션을 이용
  * sh -n 스크립트 : 문법 에러만 검사. 명령 실행 안함
  * sh -v 스크립트 : 명령 실행 전에 스크립트 내용 에코
  * sh -x 스크립트 : 명령 실행 후 명령 에코

쉘 스크립트 응용

* 과제 1

1. 사용자로 부터 “연산자 피연산자 1 피연산자 2” 를 입력하면 이를 계산하여 출력하는 쉘 스크립트 simplecal.sh 를 작성하라
연산자 : +, - 만 입력 가능
* 피연산자 : 정수만 입력 가능
* 한번에 여러 값을 입력할때 : read val1 val2 val3 와 같은 표현식 가능
2. 함수 정의를 통해 계산 기능 구현
* calculate() 함수를 정의하여 실제 사칙 연산을 수행하라
* 이 함수는 사용자로부터 입력받은 “연산자 피연산자 1 피연산자 2” 를입력받는다.
* 계산 결과를 출력한다. 출력은 “피연산자1 연산자 피연산자2 = 결과” 형식이 되도록 하라.

```
#!/bin/sh

calculate() {
    read op val1 val2
    case "$op" in
        +) result=$(($val1 + $val2));;
        -) result=$(($val1 - $val2));;
        *) echo "잘못된 인자입니다."
    esac
    echo "$val1 $op $val2 = $result"
}

calculate

exit 0
```

* 과제 2

1. 작성하는 스크립트(myshedule.sh)는 최소한 한 개의 입력 매개변수를 받아들여야 한다.
   * -date:특정날짜(YYYY-MM-DD형식)에해당하는일정을출력 o -all:저장된모든일정을출력
   * -input: 사용자로 부터 일정을 입력받아서 schedule.txt 에 추가
2. 다음의 함수를 작성하라:
   * add_schedule():사용자입력을받아일정을추가하는함수. 일정은YYYY-MM-DD 형식의 날짜와, 해당 날짜의 스케줄 내용 (영문으로 20 글자 이내) 을 입력받아서 schedule.txt 에 계속 추가하도록 하라.
   * show_schedule(date):특정날짜의일정을출력하는함수.YYYY-MM-DD 형식의날짜를입력하면해당날짜의스케줄을출력한다. 출력형식은 "YYYY-MM-DD:스케줄 내용" 으로 하라.
   * show_all_schedules():모든일정을출력하는함수.모든일정을 "YYYY-MM-DD:스케줄 내용" 형식의 반복으로 출력하라.
   * 스케줄 입력시에는 사용자가 추가할 일정이 더 이상 없을 때까지 계속 일정을 입력 받을 수 있도록 한다
4. 작성한 쉘스크립트는 다음과 같이 실행될 수 있어야 한다.
   * 일정 입력을 위한 실행 : myschedule.sh -input
   * 날짜를 지정하여 일정을 출력하는 실행 : myschedule.sh -date 2024- 05-20
   * 모든 일정을 출력하는 실행 : myschedule.sh -all

```
#!/bin/sh

add_schedule() {
    while true
    do
        read date
        if [ -z ${date} ]
            then return
        fi
        read schedule
        echo "$date : $schedule" >> schedule.txt
    done
}

show_schedule(){
    date=$1
    grep "$date" schedule.txt
    return
}

show_all_schedule(){
    cat schedule.txt
    return
}

case "$1" in
    -date) show_schedule $2;;
    -all) show_all_schedule;;
    -input) add_schedule;;
esac

exit 0
```

현재 디렉토리에서 f로 시작하는 파일이나 디렉토리에 대 해서 파일인 경우에는 “파일명 : 일반파일”, 디렉토리인 경우에는 “디렉토리명 : 디렉토리” 를 출력하도록 쉘 프로그램을 작성하시오.

```
#!/bin/bash

for ((i=1; i<=100; i++)); do
    dir_name=$(printf "user%03d" "$i")
    mkdir "$dir_name"
done

exit 0
```

`