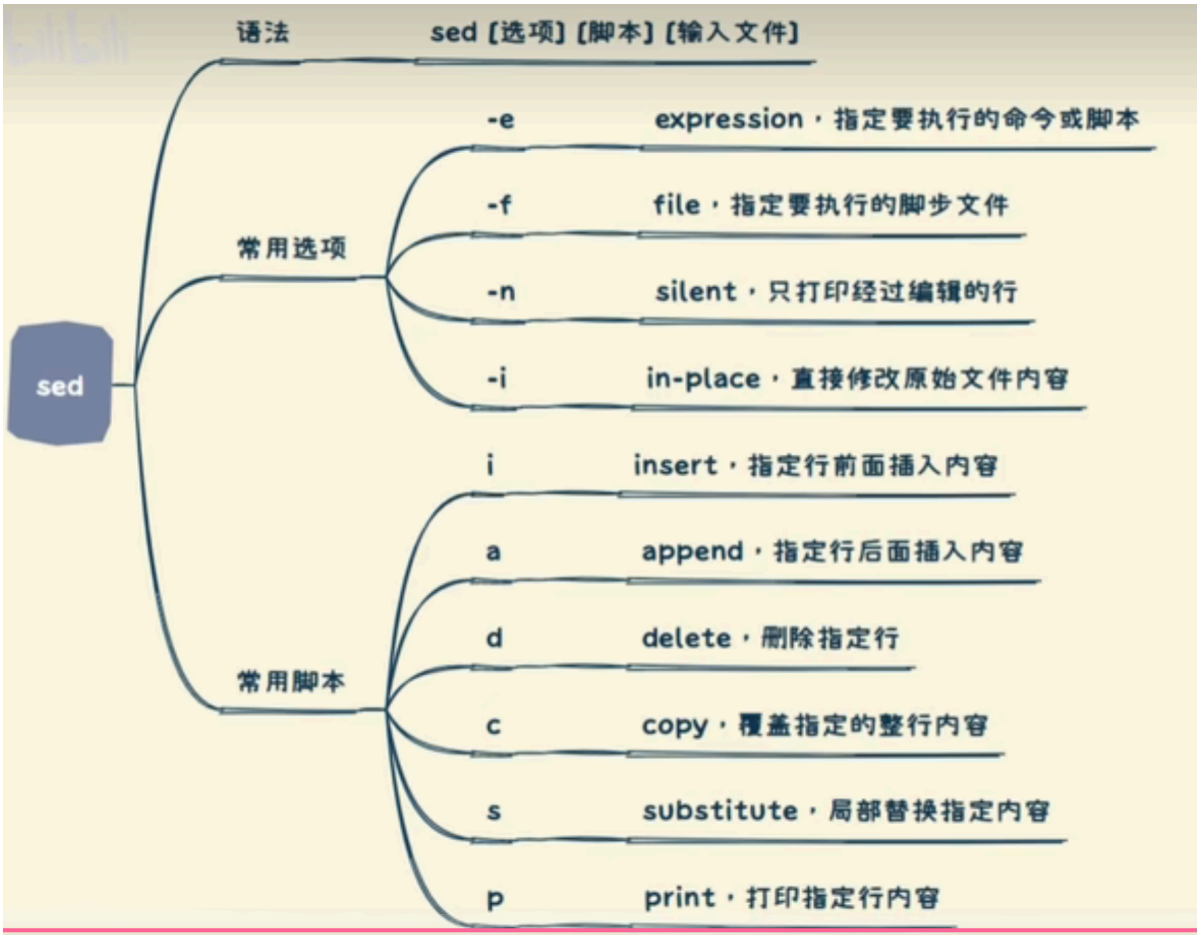
## grep


补充：grep -c "str" file　只输出匹配到的行

# sed




eg:

```
sed -n '3p' my.txt
# 输出 my.txt 的第三行
sed '2d' my.txt
# 删除 my.txt 文件的第二行
sed '2,$d' my.txt
# 删除 my.txt 文件的第二行到最后一行
sed 's/str1/str2/g' my.txt
# 在整行范围内把 str1 替换为 str2。
# 如果没有 g 标记，则只有每行第一个匹配的 str1 被替换成 str2
sed -e '4astr ' -e 's/str/aaa/' my.txt
#-e 选项允许在同一行里执行多条命令。例子的第一条是第四行后添加一个 str，
# 第二个命令是将 str 替换为 aaa。命令的执行顺序对结果有影响。
```

# awk

## 语法

```
awk options 'pattern {action}' file
```

**选项参数说明：**

- options：是一些选项，用于控制 awk 的行为。
- pattern：是用于匹配输入数据的模式。如果省略，则 awk 将对所有行进行操作。
- {action}：是在匹配到模式的行上执行的动作。如果省略，则默认动作是打印整行。

以下是一些常见的 awk 命令用法：

打印整行：

```
awk '{print}' file
```

打印特定列：

```
awk '{print $1, $2}' file
```

使用分隔符指定列：

```
awk -F',' '{print $1, $2}' file
```

打印行数：

```
awk '{print NR, $0}' file
```

打印行数满足条件的行：

```
awk '/pattern/ {print NR, $0}' file
```

计算列的总和：

```
awk '{sum += $1} END {print sum}' file
```

打印最大值：

```
awk 'max < $1 {max = $1} END {print max}' file
```

格式化输出：

```
awk '{printf "%-10s %-10s\n", $1, $2}' file
```

## Lab0 实例

- (1)

3、在 src/sh_test 目录下，有一个 file 文件和 hello_os.sh 文件。hello_os.sh 是一个未完成的脚本文档，请同学们借助 shell 编程的知识，将其补完，以实现通过命令 bash hello_os.sh AAA BBB，在 hello_os.sh 所处的目录新建一个名为 BBB 的文件，其内容为 AAA 文件的第 8、32、128、512、1024 行的内容提取 (AAA 文件行数一定超过 1024 行)。[注意：对于命令 bash hello_os.sh AAA BBB，AAA 及 BBB 可为任何合法文件的名称，例如 bash hello_os.sh file hello_os.c，若已有 hello_os.c 文件，则将其原有内容覆盖]

```
#!/bin/bash

# 获取输入参数
SOURCE_FILE=$1
TARGET_FILE=$2

# 提取指定行的内容并写入目标文件
sed -n '8p;32p;128p;512p;1024p' "$SOURCE_FILE" > "$TARGET_FILE"
```

- (2)

1、在 Lab0 工作区的 ray/sh_test2 目录下，存在一个未补全的 search.sh 文件，将其补完，以实现通过命令 bash search.sh file int result，可以在当前目录下生成 result 文件，内容为 file 文件含有 int 字符串所在的行数，即若有多行含有 int 字符串需要全部输出。[注意：对于命令 bash search.sh file int result，file 及 result 可为任何合法文件名称，int 可为任何合法字符串，若已有 result 文件，则将其原有内容覆盖，匹配时大小写不忽略 ]

```
#!/bin/bash
pattern=$2
sed -n "/$pattern/=" $1 > $3
```

```
#!/bin/bash
filea=$1
pattern=$2
fileb=$3
awk -v pat="$pattern" 'index($0, pat) {print NR}' "$filea" > "$fileb"
```

- (3)

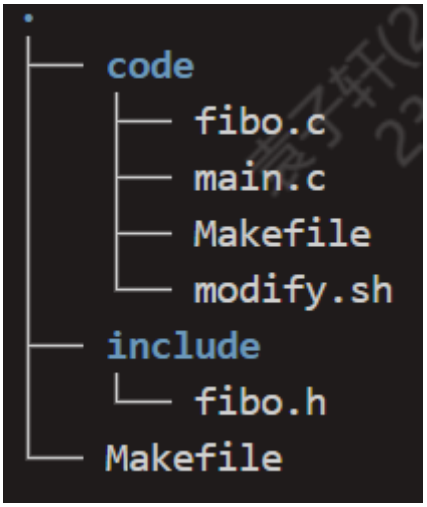
1、在 Lab0 工作区的 csc/code 目录下，存在 fibo.c、main.c，其中 fibo.c 有点小问题，还有一个未补全的 modify.sh 文件，将其补完，以实现通过命令 bash modify.sh fibo.c char int，可以将 fibo.c 中所有的 char 字符串更改为 int 字符串。[注意：对于命令 bash modify.sh fibo.c char int，fibo.c 可为任何合法文件名，char 及 int 可以是任何字符串，评测时评测 modify.sh 的正确性，而不是检查修改后 fibo.c 的正确性]

```
#!/bin/bash
FILE=$1
SRC=$2
DST=$3
sed -i "s/$SRC/$DST/g" "$FILE"
```

## Makefile

2、Lab0 工作区的 csc/code/fibo.c 成功更换字段后 (bash modify.sh fibo.c char int)，现已有 csc/Makefile 和 csc/code/Makefile，补全两个 Makefile 文件，要求在 csc 目录下通过命令 make 可在 csc/code 目录中生成 fibo.o、main.o，在 csc 目录中生成可执行文件 fibo，再输入命令 make clean 后只删除两个 .o 文件。[注意：不能修改 fibo.h 和 main.c 文件中的内容，提交的文件中 fibo.c 必须是修改后正确的 fibo.c，可执行文件 fibo 作用是输入一个整数 n(从 stdin 输入 n)，可以输出斐波那契数列前 n 项，每一项之间用空格分开。比如 n=5，输出 1 1 2 3 5]

要求成功使用脚本文件 modify.sh 修改 fibo.c，实现使用 make 命令可以生成 .o 文件和可执行文件，再使用命令 make clean 可以将 .o 文件删除，但保留 fibo 和 .c 文件。最终提交时文件中 fibo 和 .o 文件可有可无。




- 顶层Makefile:

```
.PHONY: clean
fibo: code/fibo.o code/main.o
	gcc code/main.o code/fibo.o -o fibo

code/fibo.o:
	cd code && make fibo.o

code/main.o:
	cd code && make main.o

clean:
	cd code && make clean
```

- 子目录Makefile

```
.PHONY: clean

fibo.o: fibo.c ../include/fibo.h
        gcc -I../include -c fibo.c -o fibo.o

main.o: main.c ../include/fibo.h
        gcc -I../include -c main.c -o main.o

clean:
        rm -f fibo.o main.o
```

注：对于不在同一目录下的文件，需要用`-I`指定路径

### 其他例子

```
.PHONY: clean all

all: hello world
	echo "all done"

hello: main.o message.o
	gcc main.o message.o -o hello

world: main.o message.o
	gcc main.o message.o -o world

main.o: main.c
	gcc -c main.c

message.o: message.c
	gcc -c message.c

clean:
	rm -f *.o hello world
```