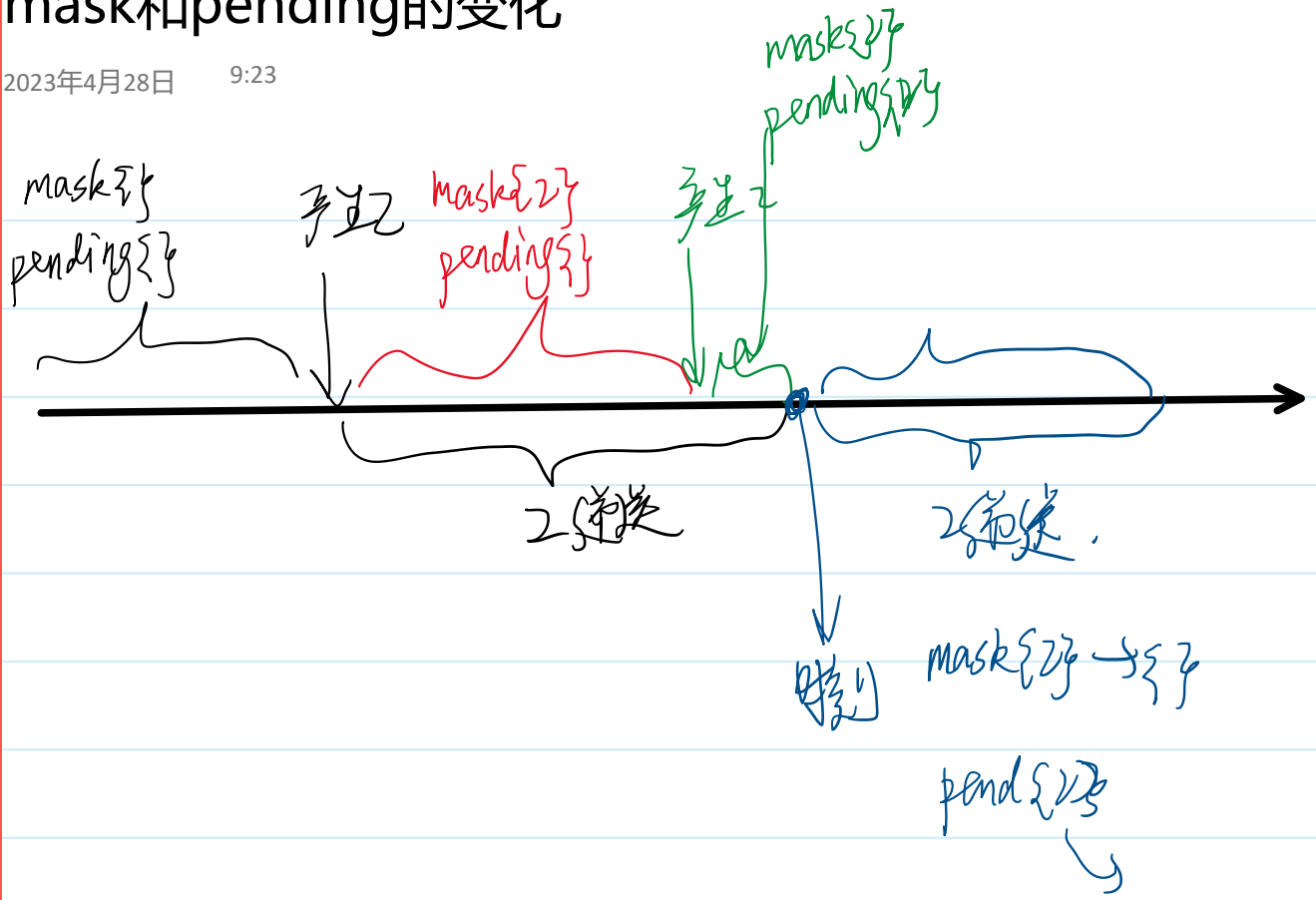
mask和pending的变化

2023年4月28日 9:23



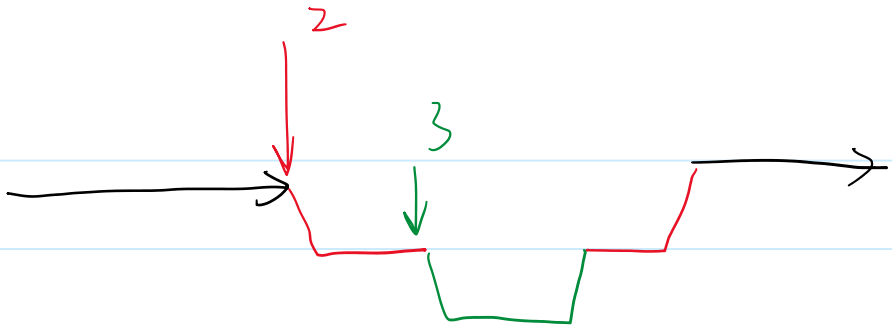
分区 新分区 9 的第 1 页

在递送2的过程中，产生一个3

不同类的信号，是不会屏蔽

2023年4月28日 11:17

```
^Cbefore, signum = 2
^\before, signum = 3
after, signum = 3
after, signum = 2
```

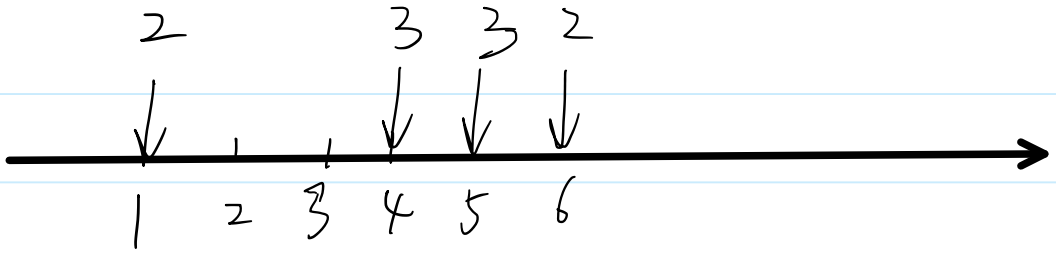


3_signal.c

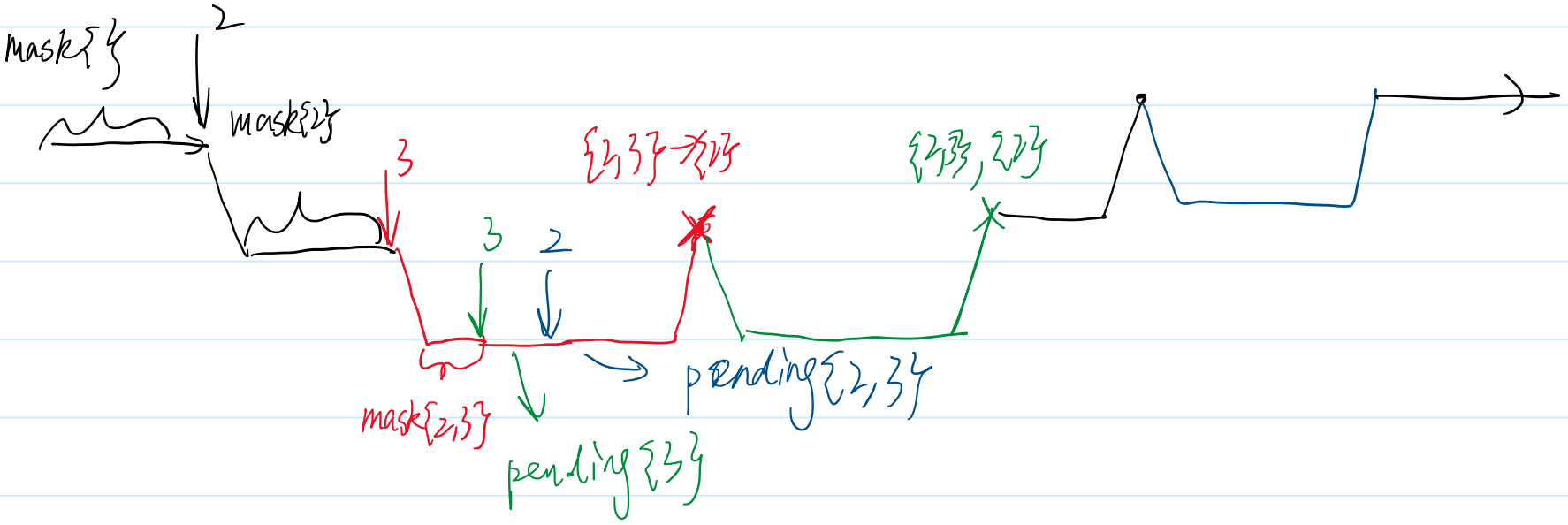
```c
#include <49func.h>
void sigFunc(int signum){
    printf("before, signum = %d\n", signum);
    sleep(10);
    printf("after, signum = %d\n", signum);
}
int main()
{
    signal(SIGINT,sigFunc);
    signal(SIGQUIT,sigFunc);
    while(1){
        sleep(1);
    }
    return 0;
}
```

比较复杂的递送情况

2023年4月28日 11:22



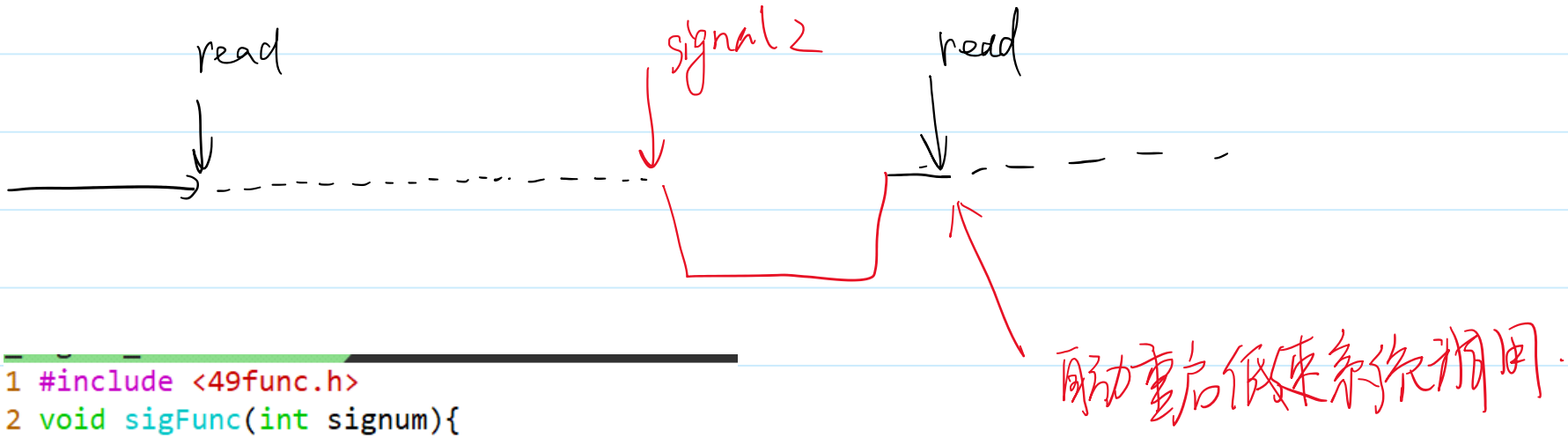
递送后私



低速系统调用

2023年4月28日 11:34

可能引发永久阻塞 read(管道, 输入)




```c
#include <49func.h>
void sigFunc(int signum){
    printf("signum = %d\n", signum);
}
int main()
{
    signal(SIGINT,sigFunc);
    char buf[4096] = {0};
    read(STDIN_FILENO,buf,sizeof(buf));
    printf("buf = %s\n", buf);
    return 0;
}

```

sigaction 注册信号

2023年4月28日 11:41

signal的特点：① 一次注册，永久生效。

② 递送信号时，临时屏蔽，不屏蔽非本。

③ 递送完成之后，自动重启低速系统调用

```
int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
```

新的递送行为

保存旧的递送行为。（NULL表示不获取旧的行为）

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

2选1，注册的行为

临时屏蔽

属性

~~void (*sa_restorer)(void);~~ X未用

sigaction

2023年4月28日 11:54

5_sigaction.c

```c
#include <49func.h>
void sigFunc(int signum){
    printf("before signum = %d\n", signum);
    sleep(5);
    printf("after signum = %d\n", signum);
}
int main()
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    act.sa_handler = sigFunc;
    sigaction(SIGINT,&act,NULL);
    sigaction(SIGQUIT,&act,NULL);
    //while(1);
    char buf[4096] = {0};
    read(STDIN_FILENO,buf,sizeof(buf));
    printf("buf = %s\n", buf);
    return 0;
}
```

一次注册 永久生效.

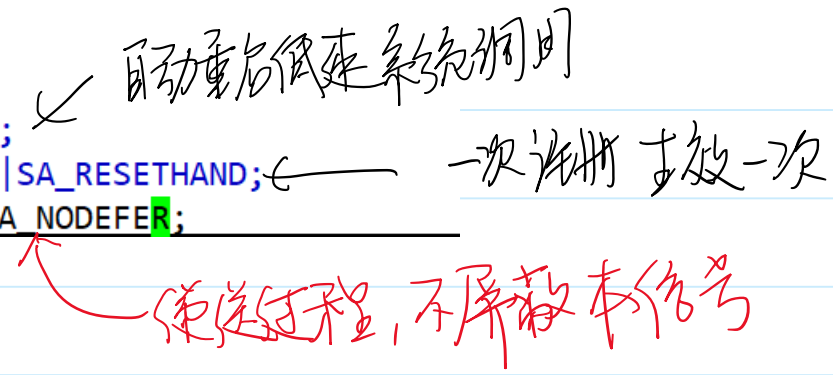
递送X时，屏蔽X

不会自动重启低速系统调用。

sigaction的属性

2023年4月28日 14:45

```
//act.sa_flags = SA_RESTART;
//act.sa_flags = SA_RESTART|SA_RESETHAND;
act.sa_flags = SA_RESTART|SA_NODEFER;
```



使用三参数版本的回调函数

2023年4月28日 14:47

① sa_flags

```
SA_SIGINFO (since Linux 2.2)
       The signal handler takes three arguments, not one.
```

② 设计一个3参数的函数.

③ 使用sa_sigaction成员.

```c
void sigFunc3(int signum, siginfo_t *info, void *p){//设计一个3参数的回调函数
    printf("signum = %d\n", signum);
    printf("pid = %d, uid = %d\n", info->si_pid, info->si_uid);
}
int main()
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    //act.sa_handler = sigFunc;
    act.sa_sigaction = sigFunc3;//使用sa_sigaction字段
    //act.sa_flags = SA_RESTART;
    //act.sa_flags = SA_RESTART|SA_RESETHAND;
    //act.sa_flags = SA_RESTART|SA_NODEFER;
    act.sa_flags = SA_RESTART|SA_SIGINFO;//使用3参数版本 SA_SIGINFO
    sigaction(SIGINT,&act,NULL);
```

临时屏蔽

2023年4月28日 14:58

内核态的进程中才屏蔽. ⟹ Mask

sigaction { 默认 屏蔽当前
SA_NODEFER. 不屏蔽

sa_mask

```
sigset_t    sa_mask;
```

位图

位图

2023年4月28日 15:02

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

每一个比特置为0

每一个比特置为1

增加一个信号

移除一个信号

判断信号是否在集合中.

2023年4月28日 15:10

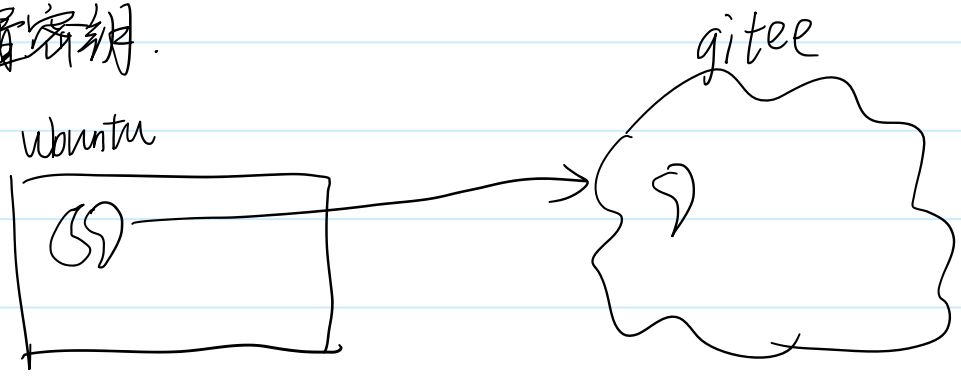
```c
void sigFunc(int signum){
    printf("before signum = %d\n", signum);
    sleep(5);
    printf("after signum = %d\n", signum);
}
int main()
{
    struct sigaction act;
    memset(&act,0,sizeof(act));
    act.sa_handler = sigFunc;
    act.sa_flags = SA_RESTART;
    sigaddset(&act.sa_mask,SIGQUIT);//递送过程中，会额外屏蔽3号信号
    sigaction(SIGINT,&act,NULL);
    char buf[4096] = {0};
    read(STDIN_FILENO,buf,sizeof(buf));
    printf("buf = %s\n", buf);
    return 0;
}
```

gitee的使用

2023年4月28日 15:17

① 注册帐号

② 配置密钥.




```
ssh-keygen
```

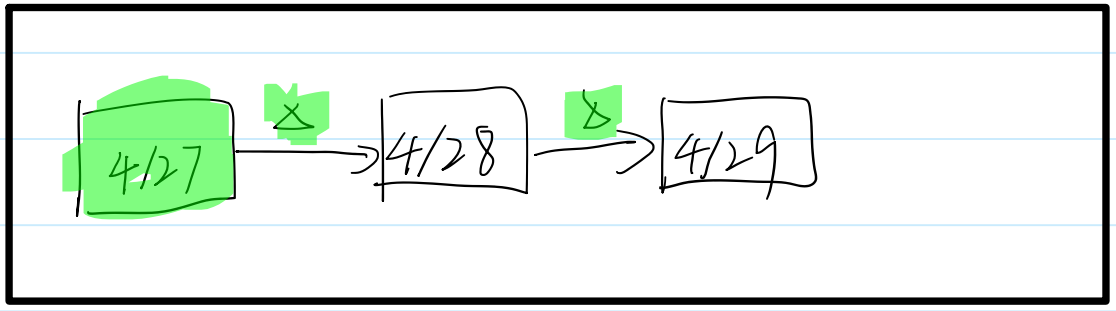
```
cd ~/.ssh/
```

```
id_rsa
id_rsa.pub
```

git

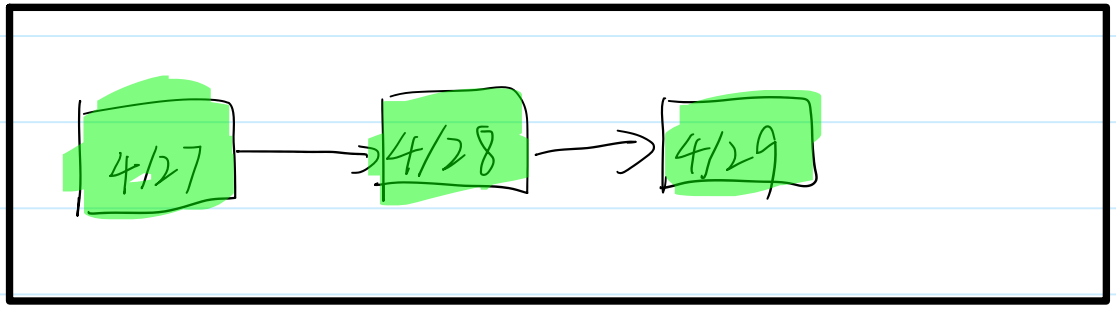
2023年4月28日 15:23

版本控制工具



SVN 工具

每一个版本存一份



git

① 只存发生变化的文件的副本

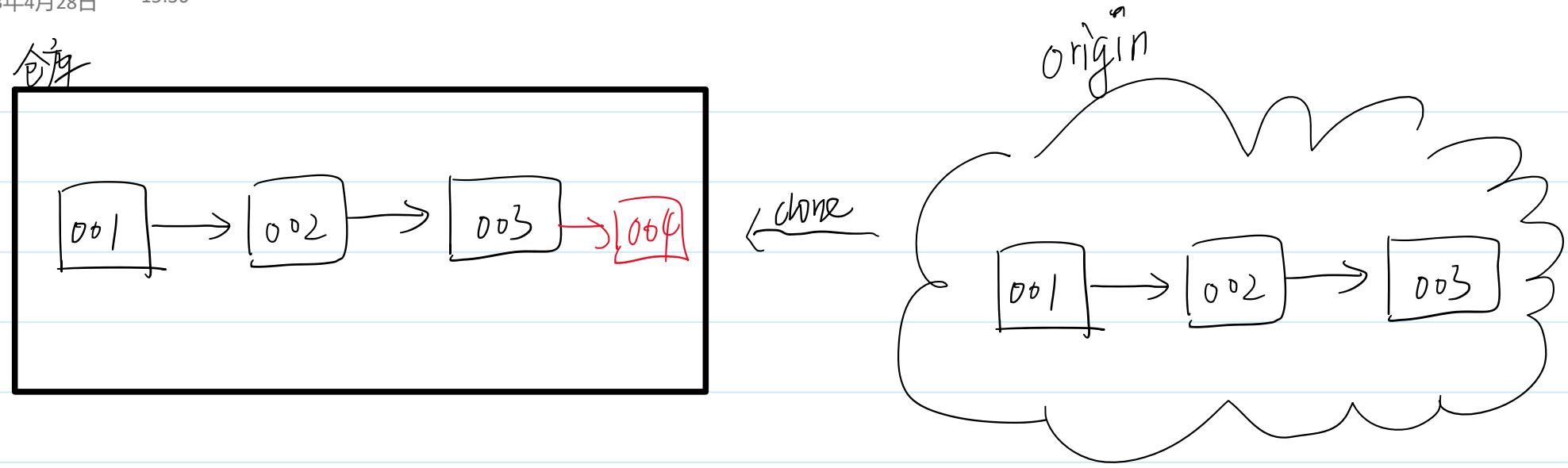
② 启用压缩.

↓

git只保留修改的文件

分布式版本管理系统

2023年4月28日 15:36



git最开始的配置

2023年4月28日 15:41

```
$ sudo apt install git

$ git config --global user.name "xxx"

$ git config --global user.email "xxx@xx.com"
```

一次需要执行一次

克隆

2023年4月28日 16:04

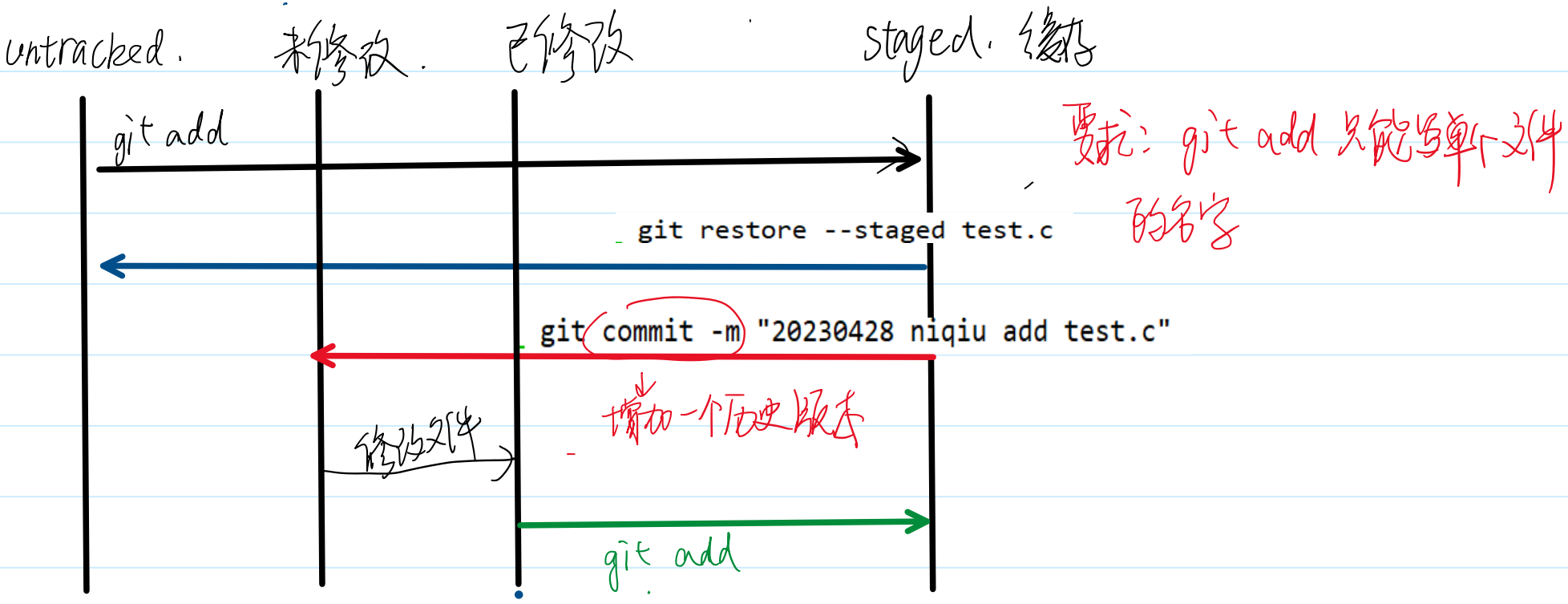
```
[liao@ubuntu ~]$ git clone git@gitee.com:liao_zheng_song/ikun48pan.git
Cloning into 'ikun48pan'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
```

```
]$ git log --all --graph --oneline
```

历史记录.

文件的状态

2023年4月28日 16:10



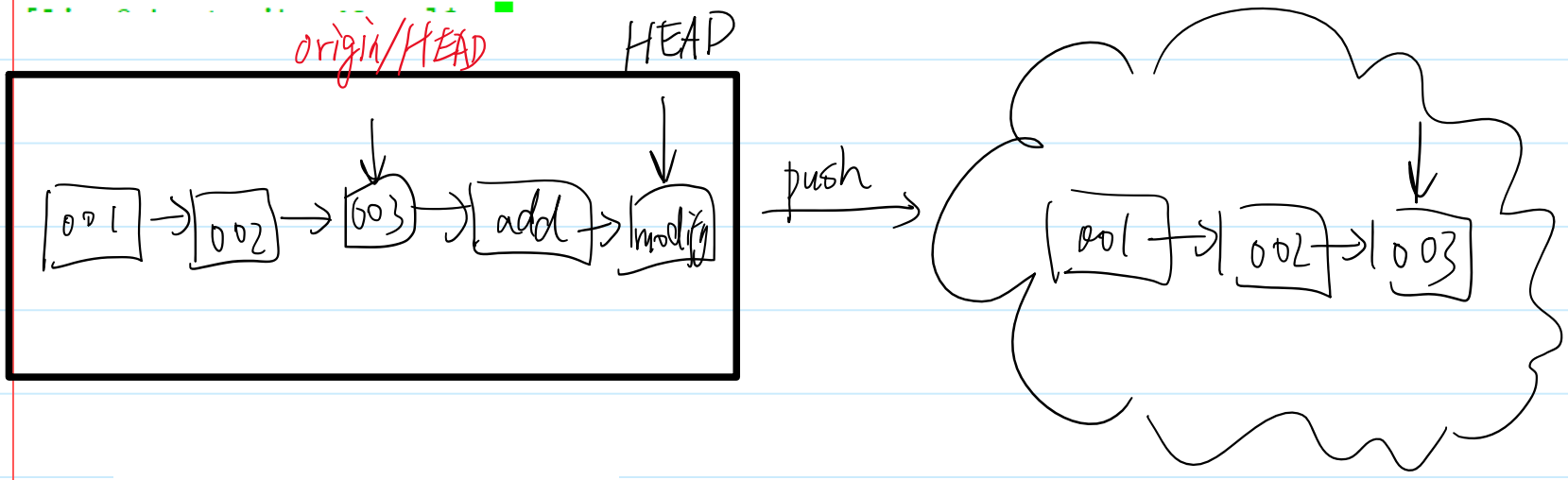
把本地的修改同步到云端

2023年4月28日 16:24

当origin/HEAD在HEAD的下边的时候

git push

```
* 1eaf859 (HEAD -> master) 20230428 niqiu modify test.c
* 839a91e 20230428 niqiu add test.c
* 5be2b48 (origin/master, origin/HEAD) 003
* af0f59f 002
* 85feb89 001
```




```
$ git push origin master
```

获取pending集合的内容

2023年4月28日 16:31

未决信号：已产生但未递送的信号。

```
int sigpending(sigset_t *set);
```

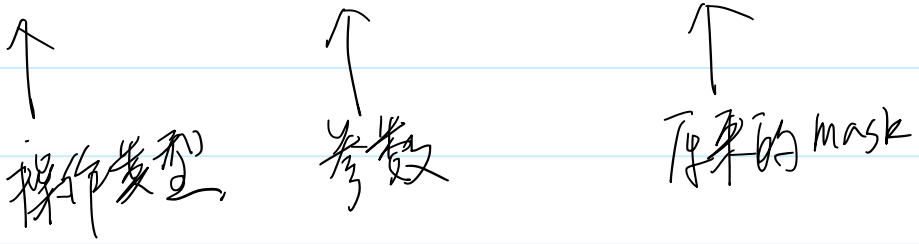
```c
#include <stdio.h>
void sigFunc(int signum){
    printf("before signum = %d\n", signum);
    sleep(5);
    // 检查一下是否存在未决的2号信号
    sigset_t pending;
    sigpending(&pending);
    if(sigismember(&pending,SIGQUIT)){
        printf("SIGQUIT is pending!\n");
    }
    else{
        printf("SIGQUIT is not pending!\n");
    }
    printf("after signum = %d\n", signum);
}
int main()
{
    signal(SIGINT,sigFunc);
    signal(SIGQUIT,sigFunc);
    while(1){
        sleep(1);
    }
    return 0;
}
```

sigprocmask

2023年4月28日 16:42 → 全程屏蔽.

signal / sigaction / sigaction 的 sa_mask ⟹ 临时屏蔽

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```



SIG_BLOCK
The s

SIG_UNBLOCK
The s
block

SIG_SETMASK

2023年4月28日 16:52

```c
int main()
{
    sigset_t set,oldset;
    sigfillset(&set);//set包含所有的信号
    sigprocmask(SIG_BLOCK,&set,&oldset);//把所有信号加入屏蔽，旧的mask存入oldset中
    printf("block all!\n");
    sleep(10);
    printf("unblock all!\n");
    sigprocmask(SIG_SETMASK,&oldset,NULL);
    while(1){
        sleep(1);
    }
    return 0;
}
```

kill raise

2023年4月28日 17:12

```
int kill(pid_t pid, int sig);
```

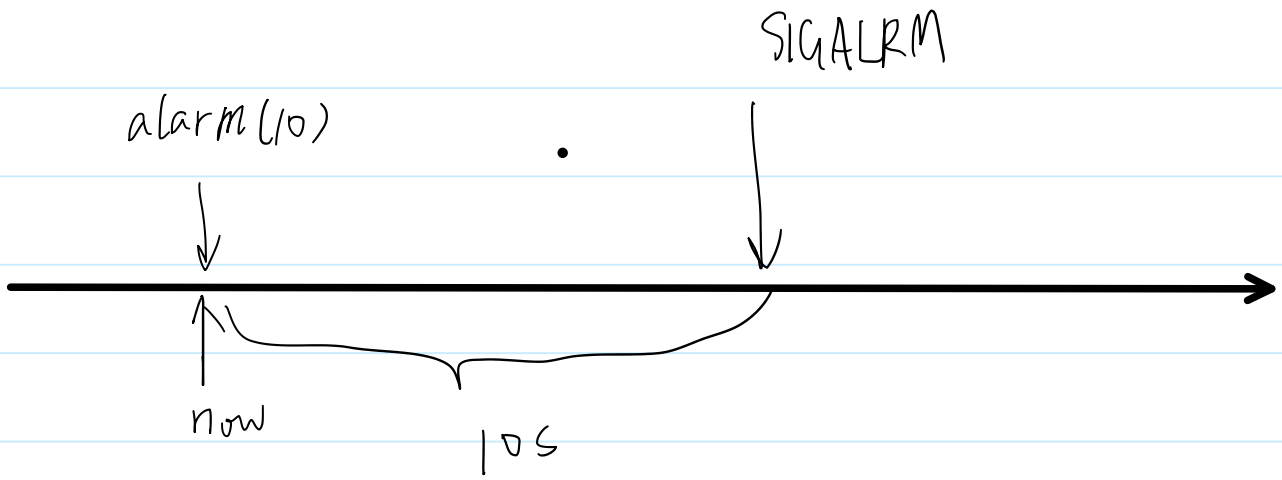
```
int main(int argc, char *argv[])
{
    // ./9_kill 12345
    ARGS_CHECK(argc,2);
    kill(atoi(argv[1]),9);
    return 0;
}
```

一般不用信号来组织业务逻辑
信号的主要功能是实现有序退出

```
int raise(int sig);
```

alarm 定闹钟

2023年4月28日 17:19

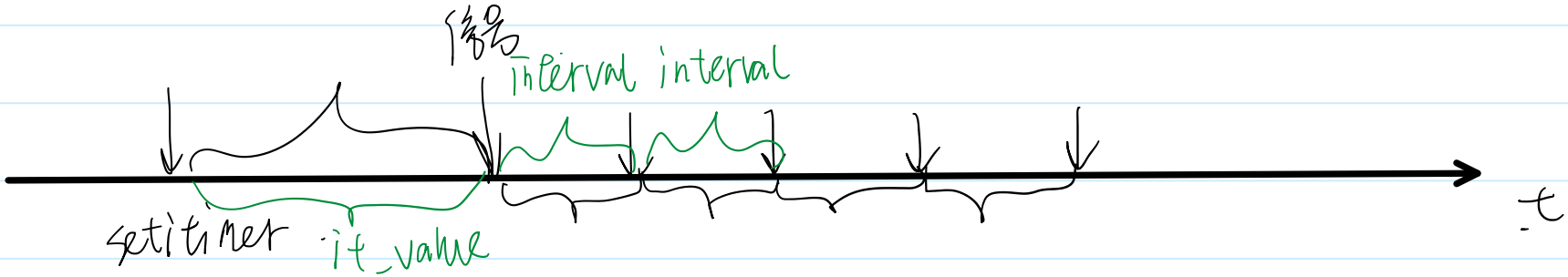



```c
void sigFunc(int signum){
    printf("signum = %d\n", signum);
}
int main()
{
    //signal(SIGINT,sigFunc);
    alarm(10);
    signal(SIGALRM,sigFunc);
    pause();//pause会在信号递送完成的时候恢复就绪
    return 0;
}
```

间隔定时器 itimer

2023年4月28日 17:29

```
int setitimer(int which, const struct itimerval *new_value,
              struct itimerval *old_value);
```



ITIMER_REAL	墙上时钟时间	SIGALRM
ITIMER_VIRTUAL	虚拟时间	the user-mode CPU time consumed by the process. SIGVTALRM
ITIMER_PROF	实用时间	SIGPROF

This timer counts down against the total (i.e., both user and system) CPU time consumed by the process.

```
struct itimerval {
    struct timeval it_interval; /* Interval for periodic timer */
    struct timeval it_value;    /* Time until next expiration */
};

struct timeval {
    time_t      tv_sec;         /* seconds */
    suseconds_t tv_usec;        /* microseconds */
};
```

定时器

2023年4月28日 17:40

```c
#include <49func.h>
void sigFunc(int signum){
    printf("signum = %d\n", signum);
    time_t now = time(NULL);
    printf("curtime = %s\n",ctime(&now));
}
int main()
{
    sigFunc(0);
    struct itimerval itimer;
    itimer.it_value.tv_sec = 3;
    itimer.it_value.tv_usec = 0;
    itimer.it_interval.tv_sec = 1;
    itimer.it_interval.tv_usec = 0;
    //signal(SIGALRM,sigFunc);
    //setitimer(ITIMER_REAL,&itimer,NULL);
    signal(SIGPROF,sigFunc);
    setitimer(ITIMER_PROF,&itimer,NULL);
    while(1);
    return 0;
}
```