
리눅스 시스템



목차

- 리눅스 설치
- 커널 컴파일 & 리눅스 시스템
- 리눅스 기본 명령어
- 디렉토리 및 파일 구조의 이해
- 리눅스 셸 (shell) 소개
- vim 에디터
- 셸 프로그래밍



- 리눅스 시작하기 -

시작과 종료

- 로그인
 - ID와 PASSWORD를 입력
- 로그아웃
 - # exit*
 - # logout*
- 시스템 종료
 - shutdown, halt, init 명령을 사용
 - # shutdown -h now*
 - # halt*
 - # init 0*

시작과 종료

■ shutdown

- # *shutdown -h +10m* : 10분후에 종료 (h: halt)
- # *shutdown -r 22:00* : 오후 10시에 재가동 (r: reboot)
- # *shutdown -c* : 진행중인 shutdown을 취소 (c: cancel)
- # *shutdown -k now* : 접속중인 사용자에게 종료 메시지 전송

init 명령어와 run level

■ run level

- 0번 – 종료 모드
- 1번 – 단일 사용자 모드 (시스템 복구 시에 사용)
- 2번 – 사용 안함
- 3번 – 다중 사용자 모드 (텍스트 로그인)
- 4번 – 사용 안함
- 5번 – 다중 사용자 모드 (X 윈도우 로그인)
- 6번 – reboot 모드

■ init

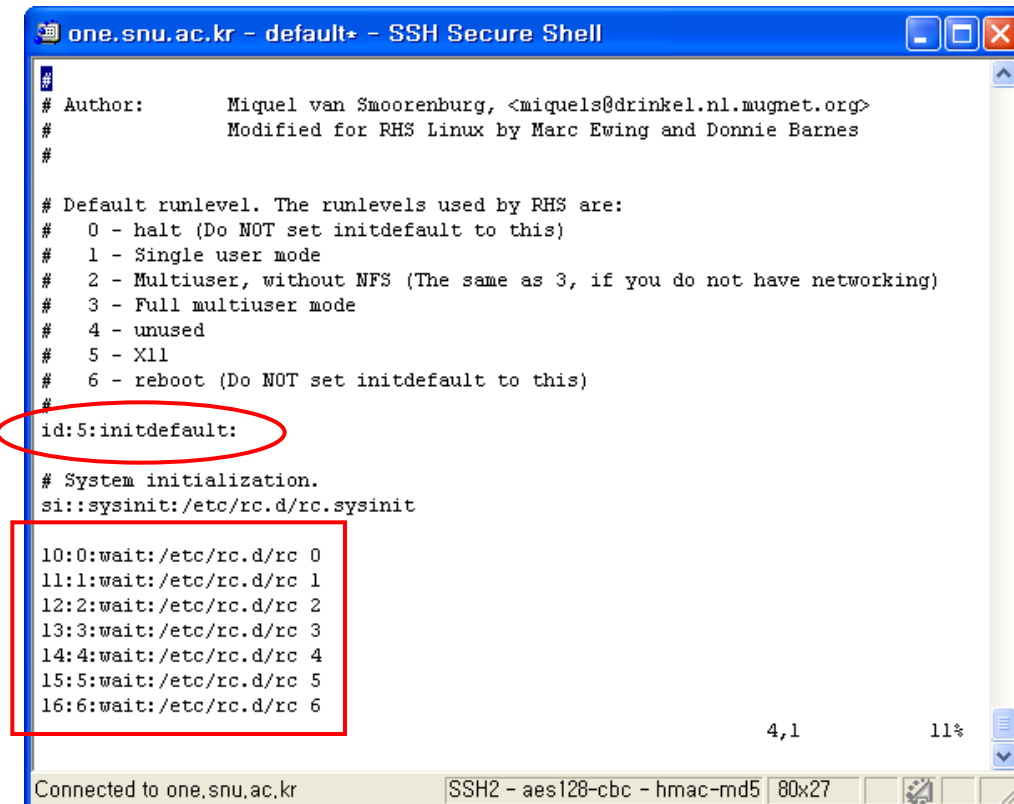
`# init 0` run level 0으로 시스템 모드를 전환

■ 부팅 시의 run level 정의 부분

- `/etc/inittab`

/etc/inittab

- 부팅 시의 run level을 정의
- run level 별로 수행 해야 할 서비스를 정의



```
#
# Author:      Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#              Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

10:0:wait:/etc/rc.d/rc 0
11:1:wait:/etc/rc.d/rc 1
12:2:wait:/etc/rc.d/rc 2
13:3:wait:/etc/rc.d/rc 3
14:4:wait:/etc/rc.d/rc 4
15:5:wait:/etc/rc.d/rc 5
16:6:wait:/etc/rc.d/rc 6
```

4,1 11%

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x27



reboot

■ reboot

shutdown -r now

reboot

init 6

로그인 관련 **script** 파일

- 로그인 시 다음의 순서대로 **script** 파일이 수행됨
 - /etc/profile
 - ~/.bash_profile
 - ~/.bashrc
 - /etc/bashrc



로그아웃 관련 **script** 파일

- 로그아웃 시 수행 되는 **script** 파일
 - ~/bash_logout

계정 추가 및 삭제

- adduser, useradd

adduser [-u UID] [-g GID] [-d HOMEDIR] [-s SHELL] <loginname>

useradd [-u UID] [-g GID] [-d HOMEDIR] [-s SHELL] <loginname>

- userdel

userdel <사용자 이름>

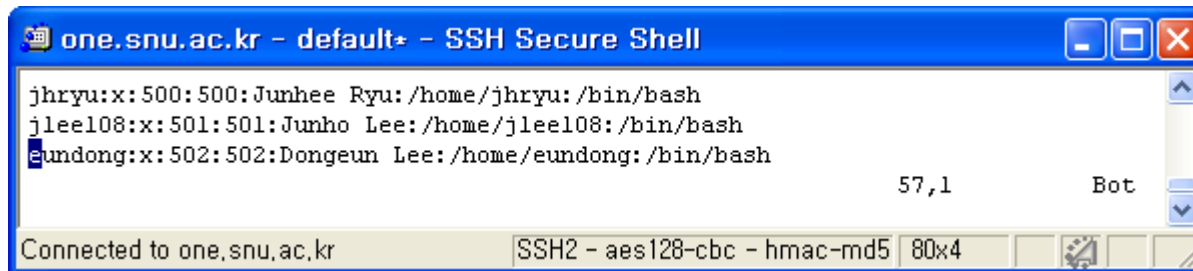
암호 변경

■ passwd

`$ passwd [사용자 이름]`

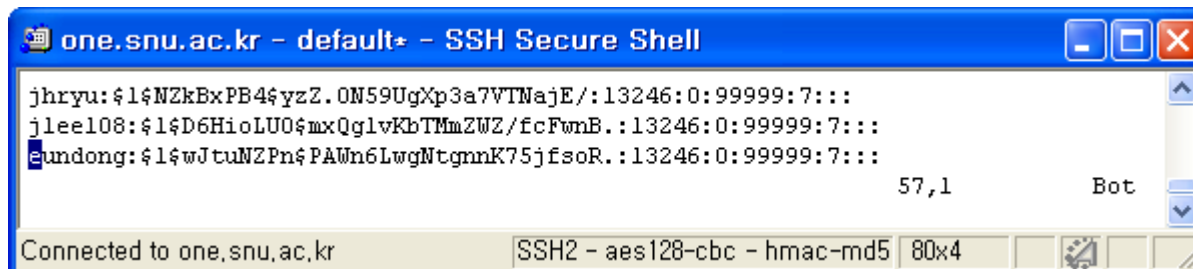
■ 관련 파일

- /etc/passwd



```
one.snu.ac.kr - default* - SSH Secure Shell
jhryu:x:500:500:Junhee Ryu:/home/jhryu:/bin/bash
jleel08:x:501:501:Junho Lee:/home/jleel08:/bin/bash
eundong:x:502:502:Donggeun Lee:/home/eundong:/bin/bash
57,1 Bot
Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4
```

- /etc/shadow



```
one.snu.ac.kr - default* - SSH Secure Shell
jhryu:$1$NZkBXpB4$yzZ.0M59UgXp3a7VTNajE/:13246:0:99999:7:::
jleel08:$1$D6HioLU0$mxQglvKbTMmZWZ/fcFwnB.:13246:0:99999:7:::
eundong:$1$wJtuNZPn$PAWn6LwgNtgmnK75jfsoR.:13246:0:99999:7:::
57,1 Bot
Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4
```

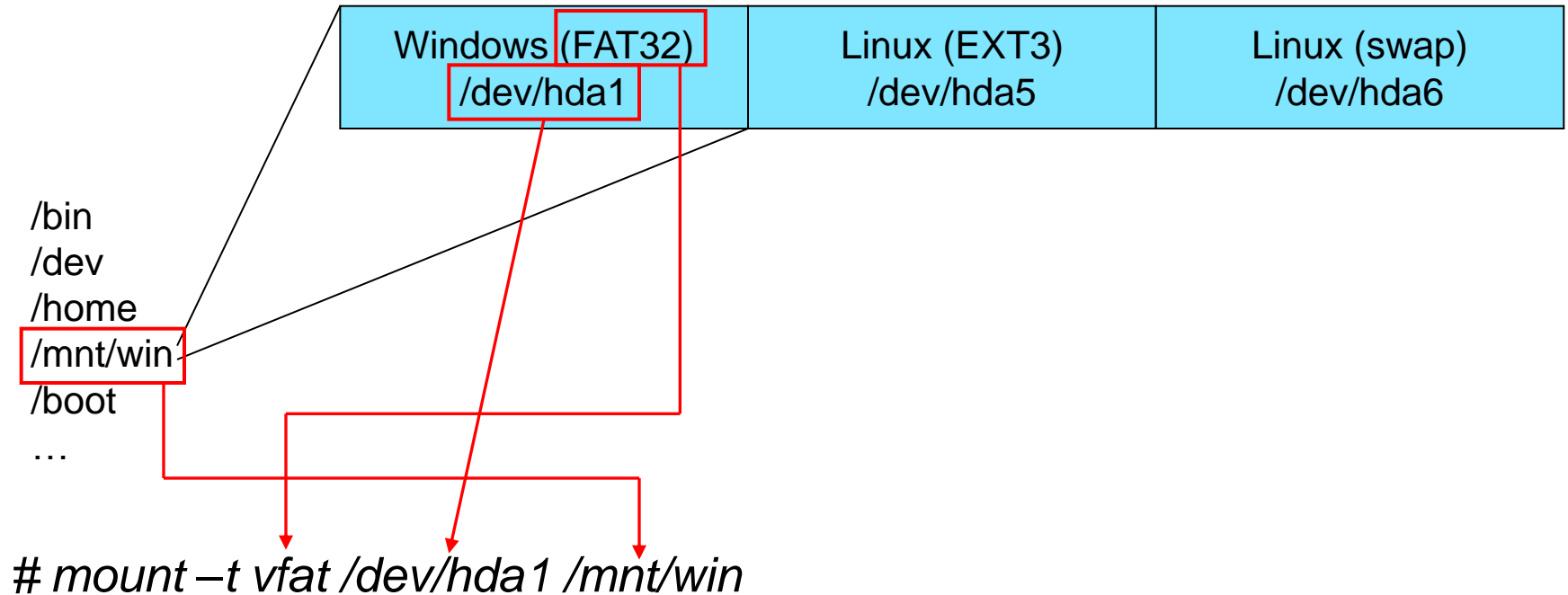
자동 완성

- 파일명의 일부만 입력한 후에 [tab] 키를 눌러 나머지 파일명을 자동으로 완성
 - /etc/sysconfig/networking/devices 디렉토리로 이동
\$ cd /et[tab]sysco[tab]networki[tab]de[tab]

마운트

■ 파티션을 특정 디렉토리에 연결하는 작업

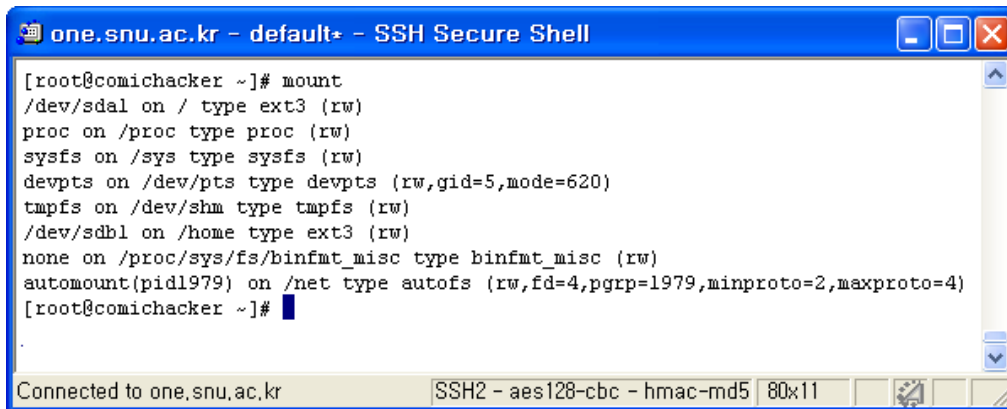
`# mount [-t type] <장치 파일> <마운트 포인트>`



마운트

■ 마운트 정보 보기

mount



```
one.snu.ac.kr - default* - SSH Secure Shell
[root@comichacker ~]# mount
/dev/sdal on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
/dev/sdbl on /home type ext3 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
automount(pid1979) on /net type autofs (rw,fd=4,pgrp=1979,minproto=2,maxproto=4)
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x11

■ 마운트 해제

umount <마운트 포인트 | 장치 파일>



- 기본 명령어 -

ls

■ 해당 디렉토리에 있는 파일의 목록을 나열

- `$ ls` : 현재 디렉토리의 파일 목록
- `$ ls /usr/bin` : /usr/bin/ 디렉토리의 목록
- `$ ls -a` : 현재 디렉토리의 목록 (hidden 파일 포함)
- `$ ls -l` : 현재 디렉토리의 목록을 자세히 보여줌
- `$ ls -l /bin/ls` : /bin/ls 파일을 자세히 보여줌
- `$ ls *.txt` : 확장자가 txt 인 목록을 보여줌
- `$ ls -l /usr/bin/a*` : /usr/bin/ 디렉토리에 있는 목록 중
앞 글자가 'a'인 것의 목록을 자세히 보여줌
- `$ ls -ld /bin` : /bin 디렉토리 파일 자체에 대해서 자세히 보여줌

cd

■ 디렉토리 이동

- `$ cd` : 현재 사용자의 홈 디렉토리로 이동
- `$ cd ~root` : `root` 사용자의 홈 디렉토리로 이동
- `$ cd ..` : 바로 상위 디렉토리로 이동
- `$ cd /usr/bin` : `/usr/bin` 디렉토리로 이동 (절대 경로)
- `$ cd ../usr/bin` : 상대 경로로 이동



pwd

- 현재 작업 디렉토리의 절대 경로를 출력

\$ pwd

: 현재 작업중인 디렉토리의 경로 출력

rm

■ 파일이나 디렉토리 삭제

- `$ rm abc.txt` : 현재 디렉토리의 `abc.txt` 파일을 삭제
- `$ rm -i abc.txt` : `abc.txt` 삭제 시 확인
- `$ rm -r abc` : `abc` 디렉토리 삭제
- `$ rm -rf abc` : `abc` 디렉토리와 그 하부를 강제로 전부 삭제

* 지우려는 파일이나 디렉토리에 삭제 권한 (w)이 있어야 함

cp

■ 파일이나 디렉토리 복사

`$ cp abc.txt cba.txt` : abc.txt 파일을 cba.txt 파일로 복사

`$ cp -r abc cba` : abc 디렉토리를 cba 디렉토리로 디렉토리 복사

* 새로 복사한 파일은 사용자의 소유가 됨

touch

- 크기가 0인 새 파일을 생성하거나 이미 존재하는 파일인 경우 수정시간을 변경

\$ touch abc.txt : 파일이 없을 경우엔 **abc.txt** 라는 빈 파일을 생성하고 **abc.txt**가 있을 경우엔 파일의 수정 시간을 현재 시각으로 변경함

mv

■ 파일과 디렉토리의 이름 변경이나 위치 이동

`$ mv aaa bbb` : aaa 파일을 bbb 디렉토리로 이동

`$ mv aaa bbb ccc ddd` : aaa, bbb, ccc 파일을 ddd 디렉토리로 이동

`$ mv abc.txt def.txt` : 이름 변경

mkdir

■ 새로운 디렉토리 생성

- `$ mkdir abc` : 현재 디렉토리 아래에 **abc** 라는 디렉토리 생성
- `$ mkdir -p def/fgh` : 현재 디렉토리 아래에 **def** 디렉토리를 생성하고 그 안에 **fgh** 디렉토리를 생성

* 생성된 디렉토리는 명령어를 수행한 사용자의 소유가 됨

rmmdir

■ 디렉토리 삭제

`$ rmmdir abc`

: 현재 디렉토리 아래의 **abc** 디렉토리 삭제

* 지우려는 디렉토리의 삭제 권한 (**w**)이 있어야 하고 디렉토리는 비어 있어야 함



cat

- 텍스트로 작성된 파일을 화면에 출력

`$ cat install.log` : install.log 파일의 내용을 화면에 출력



more

- 텍스트로 작성된 파일을 화면에 페이지 단위로 출력

`$ more install.log` : install.log 파일의 내용을 화면에 출력

* **space** 키는 다음 페이지, **b** 키는 앞 페이지로 이동



- 사용자 관리와 파일 속성 -

사용자와 그룹

- 리눅스는 다중 사용자 시스템
 - 여러 사용자가 동시에 접속해서 사용할 수 있는 시스템
- root 계정
 - root는 시스템의 모든 작업을 할 수 있는 권한이 있는 관리자
- 모든 사용자는 하나 이상의 그룹에 소속됨

사용자와 그룹

■ /etc/passwd

```
one.snu.ac.kr - default* - SSH Secure Shell
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
hsqldb:x:96:96::/var/lib/hsqldb:/sbin/nologin
tomcat:x:91:91:Tomcat:/usr/share/tomcat5:/bin/sh
beagleindex:x:58:58:User for Beagle indexing:/var/cache/beagle:/bin/false
gdm:x:42:42::/var/gdm:/sbin/nologin
jhryu:x:500:500:Junhee Ryu:/home/jhryu:/bin/bash
jleel08:x:501:501:Junho Lee:/home/jleel08:/bin/bash
eundong:x:502:502:Dongeun Lee:/home/eundong:/bin/bash
```

사용자 이름

사용자 ID

그룹 ID

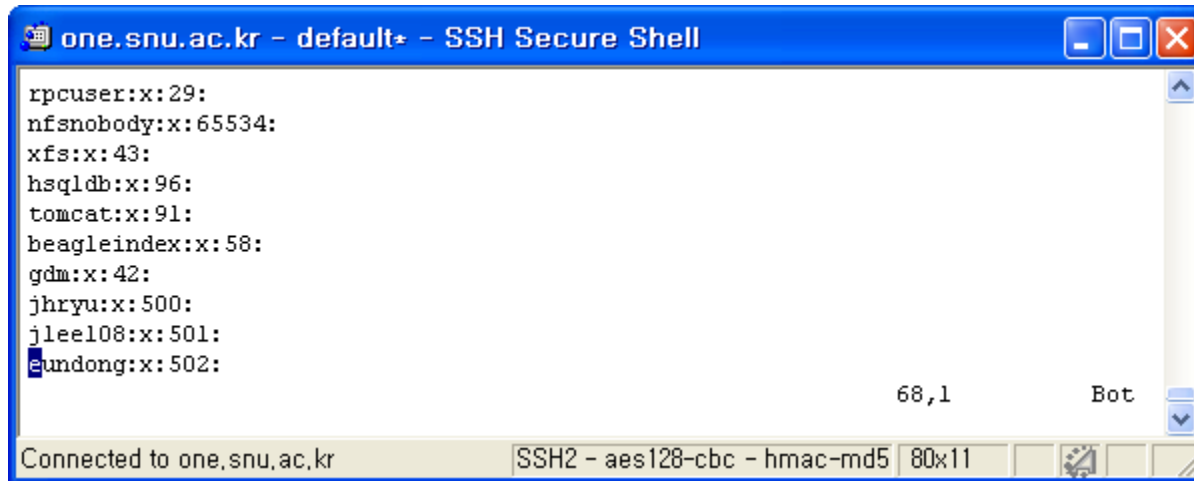
전체 이름

홈 디렉토리

로그인 셸

사용자와 그룹

■ /etc/group



The screenshot shows a terminal window titled "one.snu.ac.kr - default* - SSH Secure Shell". The terminal displays the contents of the /etc/group file, which lists system users and their associated groups. The output is as follows:

```
rpcuser:x:29:  
nfsnobody:x:65534:  
xfs:x:43:  
hsqldb:x:96:  
tomcat:x:91:  
beagleindex:x:58:  
gdm:x:42:  
jhryu:x:500:  
jlee108:x:501:  
eundong:x:502:
```

At the bottom of the terminal, there is a status bar that reads "Connected to one.snu.ac.kr" and "SSH2 - aes128-cbc - hmac-md5 80x11".

새로운 그룹 생성

■ groupadd <그룹 이름>

```
one.snu.ac.kr - default* - SSH Secure Shell
[root@comichacker ~]# groupadd newgroup
[root@comichacker ~]# adduser user1 -g newgroup
[root@comichacker ~]# adduser user2 -g newgroup
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4

■ /etc/group

```
one.snu.ac.kr - default* - SSH Secure Shell
jlee108:x:501:
eundong:x:502:
newgroup:x:503:
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4

■ /etc/passwd

```
one.snu.ac.kr - default* - SSH Secure Shell
eundong:x:502:502:Dongun Lee:/home/eundong:/bin/bash
user1:x:503:503::/home/user1:/bin/bash
user2:x:504:503::/home/user2:/bin/bash
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4

암호 변경


■ /etc/shadow



```
one.snu.ac.kr - default* - SSH Secure Shell
eundong:$1$wJtuNZPn$PAWn6LwgNtgmnK75jfsoR.:13246:0:99999:7:::
user1:!!:13253:0:99999:7:::
user2:!!:13253:0:99999:7:::
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4

■ 암호 변경



```
one.snu.ac.kr - default* - SSH Secure Shell
[root@comichacker ~]# passwd user1
Changing password for user user1.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@comichacker ~]# passwd user2
Changing password for user user2.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x11

■ /etc/shadow



```
one.snu.ac.kr - default* - SSH Secure Shell
eundong:$1$wJtuNZPn$PAWn6LwgNtgmnK75jfsoR.:13246:0:99999:7:::
user1:$1$fIaSQJ6D$Gu8iF4r6NRQ995m9ilnw/1:13253:0:99999:7:::
user2:$1$AAAtPcm$9ZxFWA61VGrWfrDwVtPkml:13253:0:99999:7:::
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x4



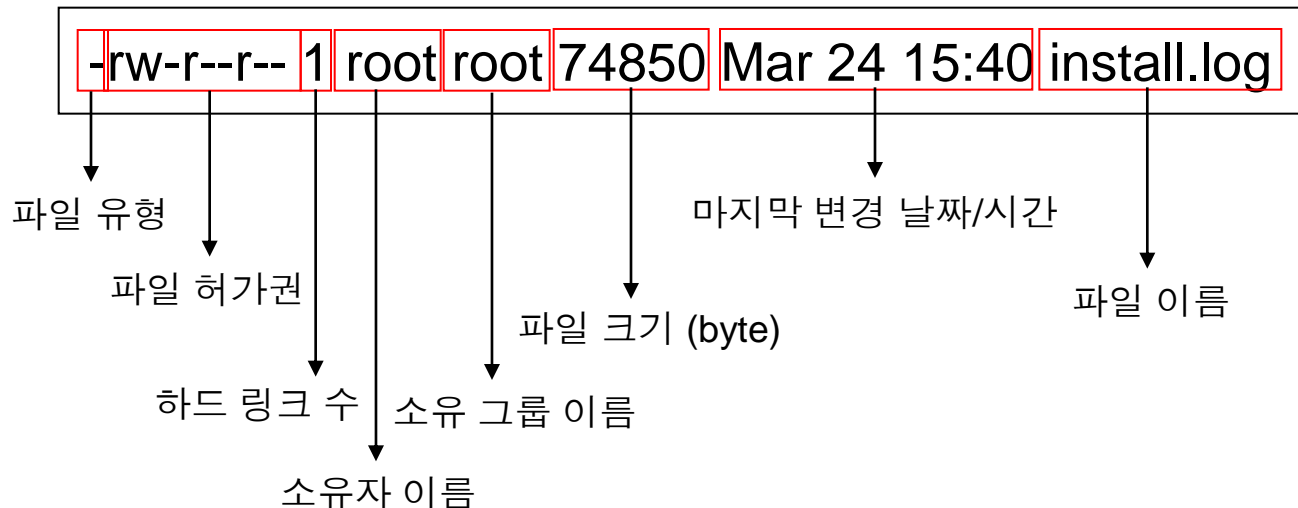
그룹 삭제

- groupdel <그룹 이름>

파일과 디렉토리의 소유와 허가권

■ ls -l

```
one.snu.ac.kr - default* - SSH Secure Shell
[root@comichacker ~]# ls -l
total 112
drwxr-xr-x 2 root root 4096 Mar 24 17:31 Desktop
-rw----- 1 root root 1985 Mar 24 15:43 anaconda-ks.cfg
-rw-r--r-- 1 root root 74850 Mar 24 15:40 install.log
-rw-r--r-- 1 root root 7571 Mar 24 15:38 install.log.syslog
[root@comichacker ~]#
```



파일과 디렉토리의 소유와 허가권

■ 파일 유형

- 이 파일이 어떤 파일인지를 나타냄
- - : 일반파일, d : 디렉토리, b : 블록 장치, c : 문자 장치,
p : named 파이프 등, l : 심볼릭 링크

파일과 디렉토리의 소유와 허가권

■ 파일 허가권 (permission)

- r : 읽기, w : 쓰기, x : 실행

소유자			소유 그룹			그 외 사용자		
r	w	-	r	-	-	r	-	-
4	2	0	4	0	0	4	0	0
6			4			4		

■ 파일의 허가권 변경

- chmod <허가권> <파일 이름>

```
$ chmod 664 install.log
```

```
$ chmod g+w install.log
```

$$\begin{pmatrix} u \\ g \\ o \\ a \end{pmatrix} + \begin{pmatrix} + \\ - \end{pmatrix} + \begin{pmatrix} r \\ w \\ x \end{pmatrix}$$

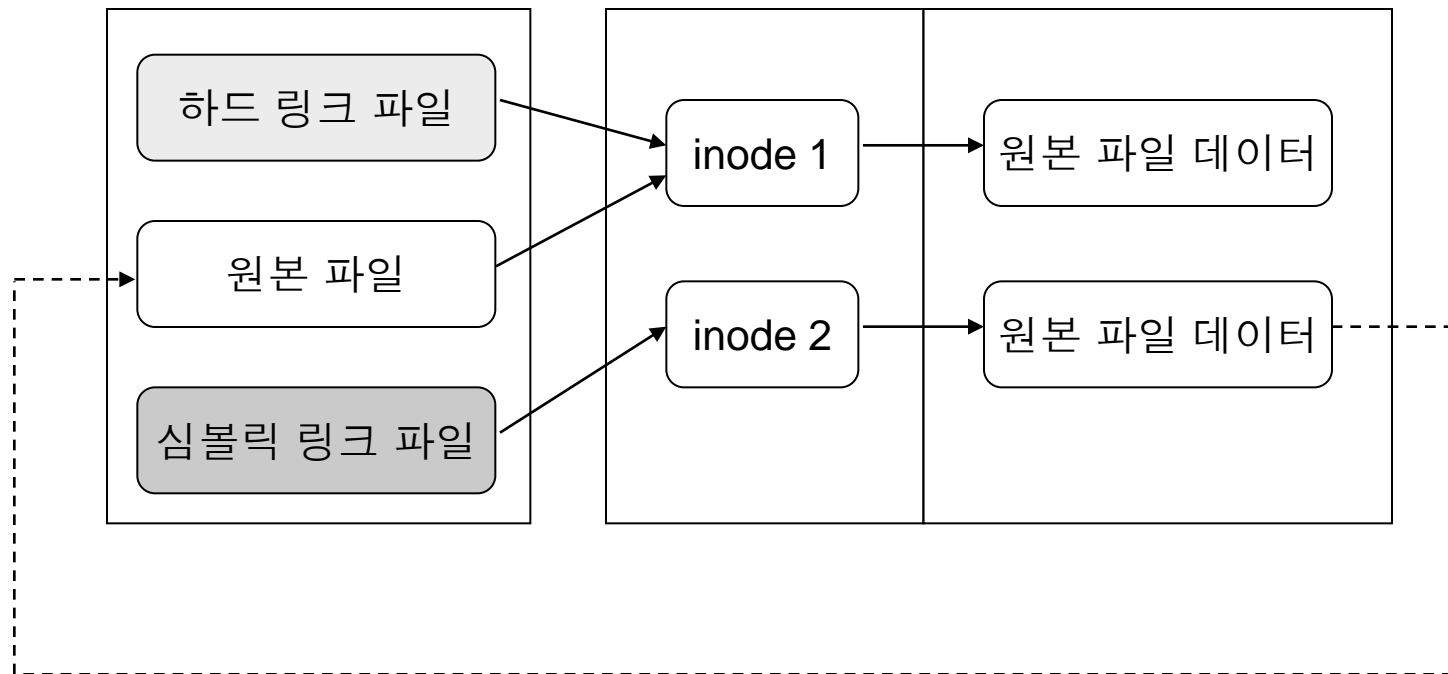
파일과 디렉토리의 소유와 허가권

- 파일 소유권
 - 파일을 소유한 사용자와 그룹을 나타냄
- 파일 소유자 변경
 - `chown [-R] <새로운 사용자 이름> <파일 이름>`
- 파일 소유그룹 변경
 - `chgrp [-R] <새로운 그룹 이름> <파일 이름>`

파일과 디렉토리의 소유와 허가권

- 링크
 - 파일을 가리키는 파일
- 링크의 종류
 - 하드 링크
 - inode를 공유하는 서로 다른 이름의 파일
 - 소프트 링크 (심볼릭 링크)
 - 가리키는 파일의 이름을 저장하고 있는 특수한 파일
- 링크의 생성
 - 하드 링크
 - `ln <링크 대상 파일명> <링크 파일명>`
 - 소프트 링크
 - `ln -s <링크 대상 파일명> <링크 파일명>`

파일과 디렉토리의 소유와 허가권





- 관리 명령어 -

파일 묶기

■ tar

● 동작

- c : 새로운 묶음
- x : 묶인 파일을 풀어줌
- t : 묶인 파일의 내용을 보여줌

● 옵션

- f : 묶음 파일명 지정, 생략 시 **tape**로 백업
- v : 파일이 묶이거나 풀리는 과정을 보여줌
- z : tar + gzip (GNU only)
- j : tar + bzip2 (GNU only)

파일 묶기

■ 예)

\$ tar cvf xinetd.tar /etc/xinetd.d/	: 생성
\$ tar cvfz xinetd.tar.gz /etc/xinetd.d/	: 생성 + gzip 압축
\$ tar cvfj xinetd.tar.bz2 /etc/xinetd.d	: 생성 + bzip2 압축
\$ tar tvf xinetd.tar	: 확인
\$ tar xvf xinetd.tar	: 풀기
\$ tar xvfz xinetd.tar.gz	: gzip 압축 해제 + tar 풀기
\$ tar xvfj xinetd.tar.gz2	: bzip2 압축 해제 + tar 풀기

파일의 압축과 풀기

■ gzip

- 압축 할 때
 - gzip <파일명> : <파일명>.gz로 압축 파일을 만들어 줌
- 압축 풀 때
 - gzip -d <gz 파일명>
 - gunzip <gz 파일명>

■ bzip2

- 압축 할 때
 - bzip2 <파일명> : <파일명>.bz2로 압축 파일을 만들어 줌
- 압축 풀 때
 - bzip2 -d <bz2 파일명>
 - bunzip2 <bz2 파일명>

파일 위치 검색

■ find [경로] [옵션] [조건] [action]

● [옵션] 설명

- name : 파일명으로 검색
- user <소유자> : 파일의 소유자로 검색
- perm <permission> : 퍼미션 값으로 검색
- mtime <날짜 수> : 수정한 날짜로 검색
- atime <날짜수> : 접근한 날짜로 검색
- newer <파일 이름> : <파일 이름> 파일이 수정된 이후 수정된 파일

● [action] 설명

- print : GNU는 기본, 유닉스는 필수 입력
- exec <command> \; : 찾은 파일로 특정 명령을 수행
- ok : exec과 같으나 명령을 수행할 지 물어봄

파일 위치 검색

■ find 사용 예)

\$ find /etc -name ".conf"*

: /etc 디렉토리 하위에 확장명이 “.conf”인 파일 검색

\$ find /home -user fedora

: /home 디렉토리 하위에 소유자가 “fedora”인 파일 검색

\$ find ~ -perm 644

: 현재 사용자의 홈 디렉토리 하위에 퍼미션이 644인 파일 검색

\$ find ~ -mtime +14 -exec rm {} \;

: 현재 사용자의 홈 디렉토리 하위에 수정한 시간이 14일이 넘은 파일을 찾아서 삭제함

파일 위치 검색

- **which** <실행 파일 이름>
 - PATH에 설정된 디렉토리만 검색
 - 절대 경로를 포함한 위치 검색
- **whereis** <실행 파일 이름>
 - 실행 파일 및 소스, man 페이지 파일까지 검색
- **locate** <파일 이름>
 - 매우 빠르고 유용하지만 새로 설치된 파일들이 등록되어 있지 않을 때는 찾을 수 없음

파일 내용 검색

■ grep

- 특정 문자열 (패턴)이 어느 파일의 어느 부분에 있는지를 검색
- `grep <찾는 내용> <파일 이름>`

■ grep 사용 예)

- `$ grep root /etc/passwd` : `/etc/passwd` 파일에서 `root`가 포함되어 있는 라인을 출력
- `$ grep -i variable *.h` : 현재 디렉토리의 헤더 파일에서 `variable` 문자열이 포함되어 있는 라인을 출력, 대소문자 구별하지 않음
- `$ grep -R printf *.c` : 현재 디렉토리 하위의 모든 `c` 소스 파일에서 `printf` 문자열이 포함되어 있는 라인을 출력

정규식

■ 정규식(Regular Expression)

- 유닉스 시스템에서 검색을 목적으로 활용
- vi, grep, ex, sed, awk, emacs, more, less

■ 메타문자

- . : 아무 문자 하나와 매치
- * : 앞의 문자가 0번 이상 나올 수 있음
- + : 앞의 문자가 1번 이상 나올 수 있음
- ^ : 라인의 시작
- \$: 라인의 끝
- [] : 스퀘어 브래킷 안의 한 문자와 매치
- \char : 메타 문자 자체를 나타냄

정규식

■ 정규식의 예

- bag : bag
- ^bag : 라인의 시작이 bag
- bag\$: 라인의 끝이 bag
- [Bb]ag : Bag 또는 bag
- b[aeiou]g : bag, beg, big, bog, bug
- b.g : b와 g사이에 아무 문자나 하나 있음
- ... : 세글자로 된 단어
- bugs* : bag, bags, bagss, bagsss ...
- bugs.* : bugs로 시작되는 모든 단어
- [A-Za-z] : 알파벳 한 문자
- [a-z]+ : 소문자로 구성된 단어

시스템 재설정

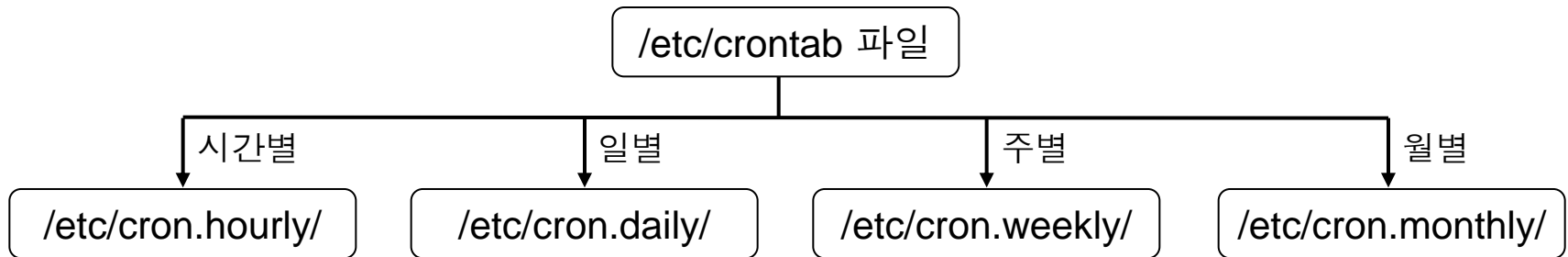
<code># system-config-display</code>	: X 윈도우 환경 설정
<code># system-config-date</code>	: 날짜 설정
<code># system-config-keyboard</code>	: 키보드 설정
<code># system-config-bind</code>	: 네임서버 설정
<code># system-config-httpd</code>	: 웹 서버 설정
<code># system-config-language</code>	: 언어 설정
<code># system-config-lvm</code>	: LVM 설정
<code># system-config-mouse</code>	: 마우스 설정
<code># system-config-network</code>	: 네트워크 환경 설정

시스템 재설정

<i># system-config-nfs</i>	: NFS 서버 설정
<i># system-config-packages</i>	: 패키지 추가 설치
<i># system-config-printer</i>	: 프린터 설정
<i># system-config-rootpassword</i>	: root 비밀번호 관리
<i># system-config-samba</i>	: samba 서버 설정
<i># system-config-securitylevel</i>	: 보안 수준 설정
<i># system-config-soundcard</i>	: 사운드 카드 설정
<i># system-config-users</i>	: 사용자 관리

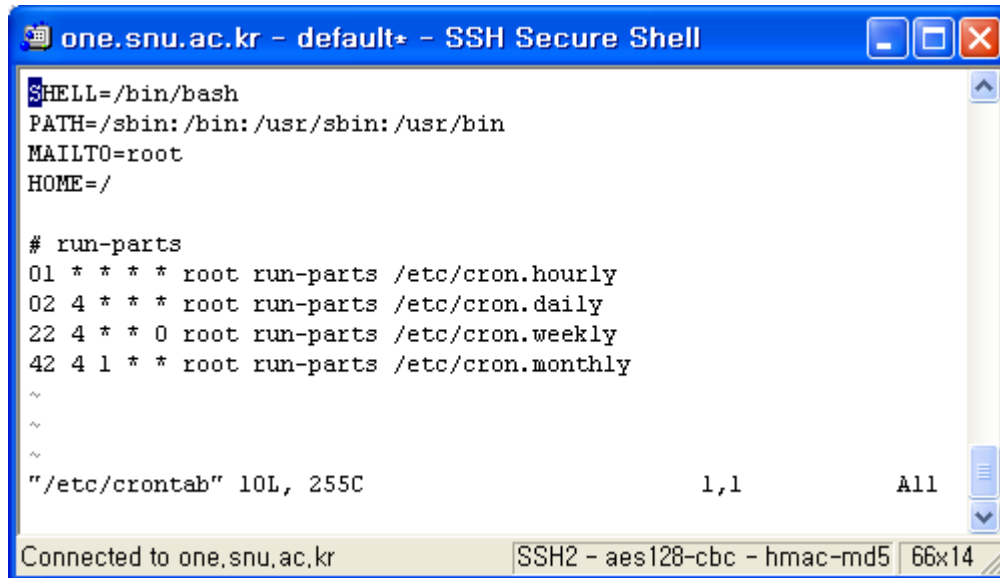
cron

- 주기적으로 반복되는 일을 자동적으로 실행될 수 있도록 설정
- 서비스 데몬 : `crond`, 설정 파일 : `/etc/crontab`



cron

■ /etc/crontab



```
one.snu.ac.kr - default* - SSH Secure Shell
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
~
~
~
"/etc/crontab" 10L, 255C          1,1          All
Connected to one.snu.ac.kr      SSH2 - aes128-cbc - hmac-md5 66x14
```

- 각 라인은 분, 시, 일, 월, 요일, 권한, 실행 명령으로 구성됨

at

- cron은 주기적으로 반복되는 작업을 예약하지만 at는 일회성 작업을 예약할 때 사용
- 예약
 - \$ at <시간>
 - \$ at 3:00am tomorrow : 내일 새벽 3시
 - \$ at 11:00pm January 30 : 1월 30일 오후 11시
 - \$ at now +1 hours : 1시간 후
 - at> 프롬프트에 예약 명령어 입력 후 [CTRL+D]
- 확인
 - \$ at -l
- 취소
 - \$ atrm <작업 번호>



- 네트워크 관련 설정 및 명령어 -

네트워크 관련 필수 개념

■ IP 주소

- 각 컴퓨터의 네트워크 카드에 부여되는 중복되지 않는 유일한 주소
- 4바이트로 이루어져 있으며, 각 자리는 0~255까지의 숫자가 올 수 있음

■ 네트워크 주소

- 같은 네트워크에 속해 있는 공통된 주소

■ 브로드캐스트 (broadcast) 주소

- 내부 네트워크의 모든 컴퓨터가 듣게 되는 주소
- 현재 주소의 제일 끝자리를 255로 바꾼 주소 (C 클래스의 경우)

네트워크 관련 필수 개념

- 게이트웨이 (gateway), 라우터 (router)
 - 네트워크 간에 데이터를 전송하는 컴퓨터 또는 장비
 - 게이트웨이를 별도로 추가하기 위한 명령어
 - `route add default gw <게이트웨이 주소> dev <장치 이름>`
- 넷마스크 (netmask), 클래스 (class)
 - 넷마스크는 네트워크의 규모를 결정 (routing에 사용)
 - C 클래스의 경우 넷 마스크는 255.255.255.0
- DNS (Domain Name Server) 서버 (=네임서버) 주소
 - URL을 해당 컴퓨터의 IP 주소로 변환해 주는 서버
 - 설정 파일 : `/etc/resolv.conf`
- 네트워크 장치 이름
 - `eth0, eth1, eth2...`

중요한 네트워크 관련 명령어

system-config-network

- DHCP 클라이언트 또는 고정 IP 주소 사용 결정
- IP 주소, 서브넷 마스크, 게이트웨이 정보 입력
- DNS 정보 입력
- 네트워크 카드 드라이버 설정
- 네트워크 장치 (eth0) 설정

system-config-network-tui

- *system-config-network*와 같은 명령어지만 텍스트 기반으로 작동

service network restart

- 네트워크 설정을 변경한 후에, 변경된 내용을 시스템에 적용

중요한 네트워크 관련 명령어

ifconfig [장치 이름]

- 해당하는 장치의 IP 주소 설정 정보를 출력해 줌
- 네트워크 장치에 IP주소, 넷마스크 등 설정이 가능하고 네트워크 장치를 활성화 또는 비활성화 할 때 사용

nslookup

- DNS 서버의 작동을 테스트하는 명령어

ping <IP 주소 또는 URL>

- 해당 컴퓨터가 네트워크에서 응답하는지를 테스트

네트워크 설정과 관련된 중요 파일

- `/etc/sysconfig/network`
 - 네트워크의 기본적인 정보가 설정되어 있는 파일
- `/etc/sysconfig/network-scripts/ifcfg-eth0`
 - `eth0` 장치에 설정된 네트워크 정보가 모두 들어있는 파일
- `/etc/resolv.conf`
 - DNS 서버 정보 및 호스트 이름이 들어 있는 파일



- 파이프, 필터, 리다이렉션 -

파이프 (pipe)

- 두 프로그램을 연결해 주는 연결 통로의 의미
- “|” 문자를 사용함
- 파이프 문자 앞의 명령어의 표준 출력을 파이프 뒤의 명령어의 표준 입력으로 전달해 줌
- 예)
`$ ls -l /etc | more`
 - “ls -l /etc” 명령을 입력하면 파일이 너무 많아서 페이지가 넘어가므로 페이지가 넘어갈 때마다 키 입력을 받음

필터 (filter)

- 필요한 것만 걸러주는 명령어
- `grep`, `tail`, `wc`, `sort`, `awk`, `sed` 등
- 주로 파이프와 같이 사용됨
- 예)
 - `$ ps -ef | grep bash`
 - “`ps -ef`”는 모든 프로세스에 대해서 출력하므로, “`bash`” 문자열이 들어간 프로세스만 출력
 - `$ rpm -qa | grep bind`
 - 설치된 패키지 중에서 “`bind`” 라는 문자열이 들어간 패키지만 출력
 - `$ ls -l /bin | sort | more`
 - “`ls -l /bin`” 결과를 사전 순으로 출력하고 화면이 넘어갈 때 키값을 입력 받음

리다이렉션 (redirection)

- 표준 입력, 출력, 에러의 방향을 바꿔줌
- 표준 입력은 키보드, 표준 출력 및 에러는 모니터이지만 이를 파일로 처리하고 싶을 때 사용
- 예)
 - `$ ls -l > aa.lst`
 - “ls -l”의 결과를 aa.lst 파일에 출력
 - `$ ls -l >> aa.lst`
 - “ls -l”의 결과를 aa.lst 파일에 추가
 - `$ sort < aa.lst`
 - aa.lst의 내용을 사전 순으로 출력
 - `$ sort < aa.lst > bb.lst`
 - aa.lst의 내용을 사전 순으로 bb.lst 파일에 출력
 - `$ gcc src.c &> error`
 - src.c 파일을 컴파일 하는데 이 과정에 발생한 에러를 error 파일에 출력



- 프로세스, 데몬, 서비스 -

프로세스

- 하드디스크에 저장된 실행 코드 (프로그램)가 메모리에 로딩되어 활성화된 것
- 용어 정리
 - foreground 프로세스
 - 사용자와 상호 작용을 하는 프로세스 (터미널을 현재 사용)
 - background 프로세스
 - 실행은 되었지만 사용자와 상호 작용을 하지 않고 있는 프로세스
 - 프로세스 번호
 - 프로세스를 구분하기 위한 고유 번호
 - 작업 번호
 - 현재 실행되고 있는 백그라운드 프로세스의 순차 번호
 - 부모 프로세스와 자식 프로세스
 - 부모 프로세스는 `fork()`를 호출한 프로세스, 자식 프로세스는 `fork()`를 통해 생성된 프로세스를 말함

프로세스 관련 명령어

■ ps

- 현재 프로세스의 상태를 확인

`$ ps` : 현재 사용자가 생성한 프로세스 정보를 출력

`$ ps -a` : 모든 사용자가 생성한 프로세스 정보를 출력

`$ ps -aux` : 데몬 프로세스 정보도 출력

■ kill

- 특정 프로세스에 특정 시그널을 전송

`$ kill 30312` : 30312번 프로세스에 종료 시그널 전달

`$ kill -SIGSTOP 30312` : 30312번 프로세스에 정지 시그널 전달

■ pstree

- 부모 프로세스와 자식 프로세스의 관계를 트리 형태로 보여줌



데몬

- 서비스 (**service**)라고도 불리는 서버 프로세스
 - 웹 서버, 네임 서버, **DB** 서버 등
- 터미널 (세션)과 단절된 프로세스

standalone 타입의 데몬

- 시스템에 독자적으로 프로세스가 구동되어 서비스를 제공
 - 웹 서버 (httpd), DB 서버 (mysqld), 메일 서버 (sendmail) 등
- 실행, 종료 및 재개는 “service <데몬 이름> <명령>”
 - # *service sshd start*
 - # */etc/init.d/network stop*
 - # */etc/init.d/sshd restart*
- /etc/init.d/*
 - standalone 타입 데몬의 실행 스크립트 파일

xinetd 타입의 데몬

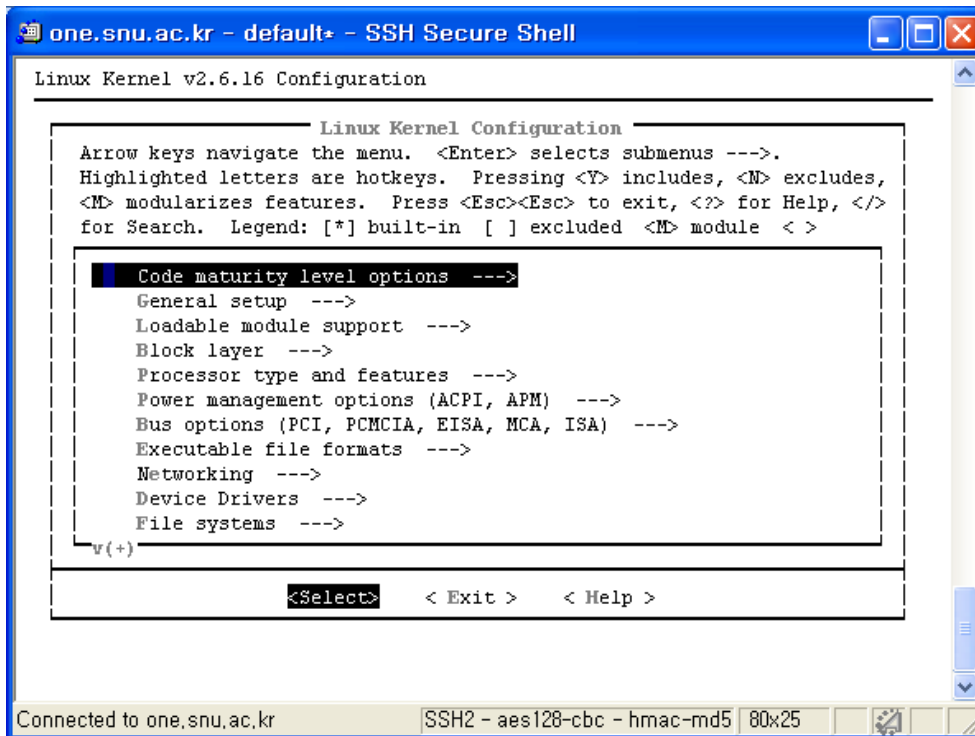
- xinetd 수퍼 데몬에 의해서 서비스 되는 데몬들을 지칭
- 요청 포트 번호에 따라 적절한 데몬과 연결 시켜줌
- 연결되는 시간은 **standalone** 타입보다 오래 걸리지만 자원 낭비를 막을 수 있음
- /etc/xinetd.d/*
 - xinetd 타입의 데몬 설정 파일
- /etc/services
 - xinetd 데몬이 서비스하는 포트의 설정 파일



- 커널 컴파일과 **GRUB** -

커널 컴파일

■ make menuconfig (config, xconfig)



- 커널에 포함시킬 구성요소들을 설정
- [*] -> 커널이 정적 포함
- [] -> 커널에 포함시키지 않음
- [M] -> 모듈로 컴파일

커널 컴파일

■ make clean (mrproper)

- 이전 컴파일 때 생성되었던 **object** 파일들을 삭제 (mrproper는 .config와 같은 설정 파일도 삭제함)

■ make dep

- 커널 구성요소들의 의존성 정보를 설정함

■ make bzImage

- 설정된 내용을 바탕으로 커널 이미지를 생성

■ make install

- 생성된 커널 이미지를 **/boot** 디렉토리에 복사하고 부트로더를 자동으로 수정해 줌

커널 컴파일

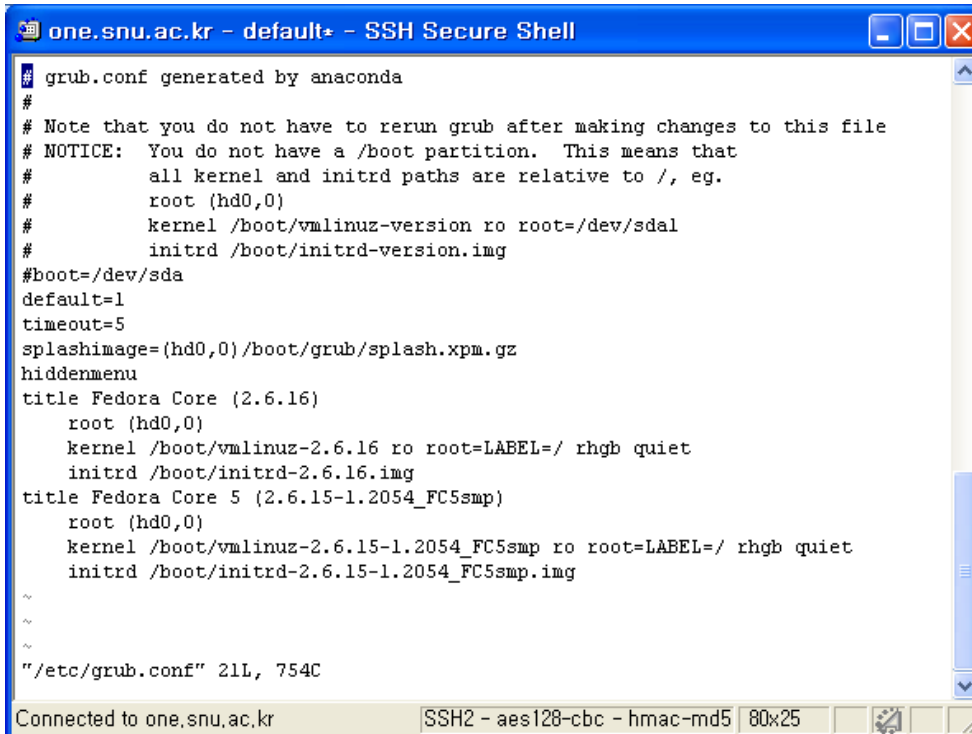
■ make modules

- module로 설정한 컴포넌트들을 모듈로 컴파일

■ make modules_install

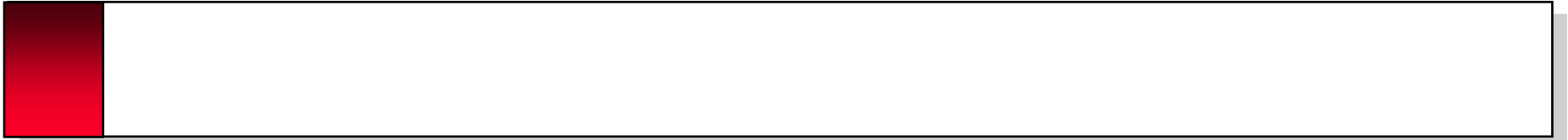
- 컴파일된 모듈들을 생성된 커널의 모듈 설치 디렉토리로 복사 (/lib/modules/`uname -r`)

GRUB



```
one.snu.ac.kr - default* - SSH Secure Shell
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE: You do not have a /boot partition. This means that
#         all kernel and initrd paths are relative to /, eg.
#         root (hd0,0)
#         kernel /boot/vmlinuz-version ro root=/dev/sda1
#         initrd /boot/initrd-version.img
#boot=/dev/sda
default=1
timeout=5
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title Fedora Core (2.6.16)
    root (hd0,0)
    kernel /boot/vmlinuz-2.6.16 ro root=LABEL=/ rhgb quiet
    initrd /boot/initrd-2.6.16.img
title Fedora Core 5 (2.6.15-1.2054_FC5smp)
    root (hd0,0)
    kernel /boot/vmlinuz-2.6.15-1.2054_FC5smp ro root=LABEL=/ rhgb quiet
    initrd /boot/initrd-2.6.15-1.2054_FC5smp.img
~
~
~
"/etc/grub.conf" 21L, 754C
Connected to one.snu.ac.kr  SSH2 - aes128-cbc - hmac-md5  80x25
```

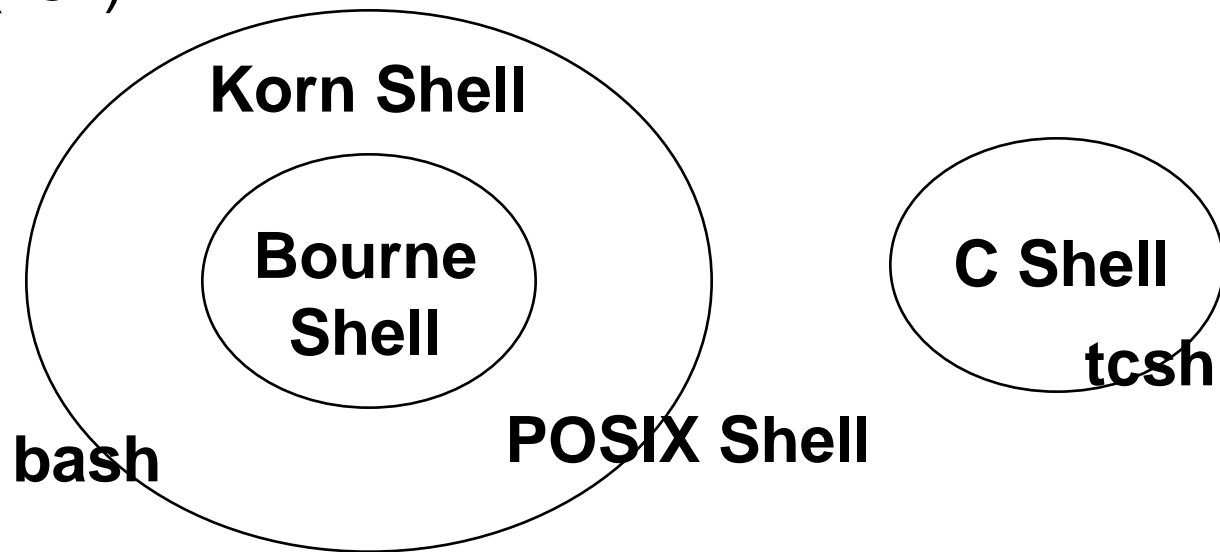
- 현재 x86에서 가장 많이 쓰이는 부트로더
- grub 상에서 부팅 옵션을 변경할 수 있음
 - e 키 : edit
 - b 키 : boot
- /etc/grub.conf
 - grub의 설정 파일



- bash 셸 -

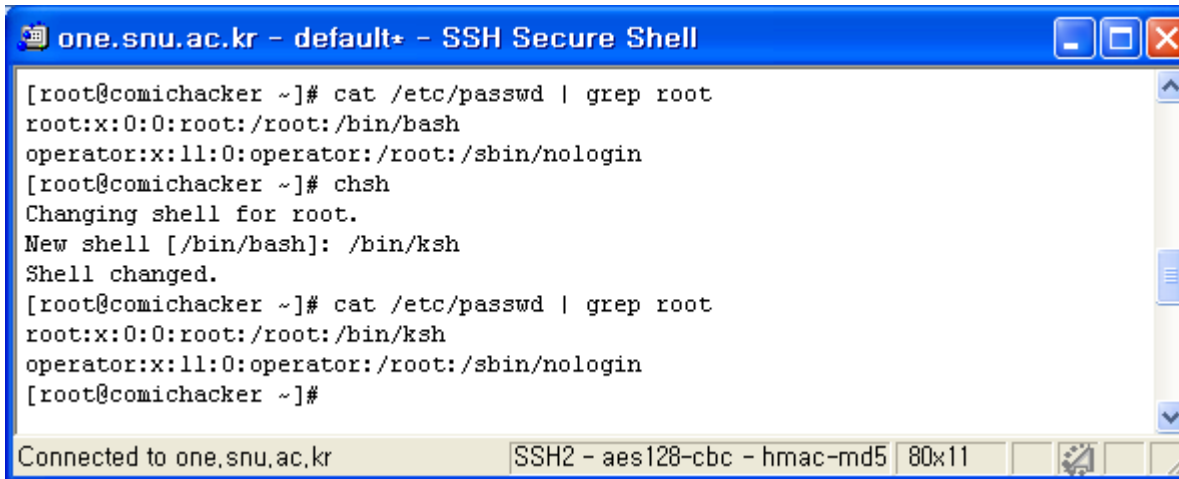
리눅스 쉘 소개

- 사용자의 명령을 처리해주는 명령어 번역기
 - Bourne Shell(sh) → Bourne Again Shell(bash)
 - Korn Shell(ksh)
 - C Shell(csh) → tcsh
 - Z Shell(zsh)



리눅스 쉘 소개

- 사람의 취향에 따라 바꿀 수 있음
 - chsh 명령어



```
one.snu.ac.kr - default* - SSH Secure Shell

[root@comichacker ~]# cat /etc/passwd | grep root
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@comichacker ~]# chsh
Changing shell for root.
New shell [/bin/bash]: /bin/ksh
Shell changed.
[root@comichacker ~]# cat /etc/passwd | grep root
root:x:0:0:root:/root:/bin/ksh
operator:x:11:0:operator:/root:/sbin/nologin
[root@comichacker ~]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 80x11

명령어의 수행

■ 단일 명령어 수행

- command

- 예)

\$ pwd

\$ wc -c /etc/passwd

\$ mkdir jhryu

■ 다중 명령어 수행

- command1; command2; ... commandn

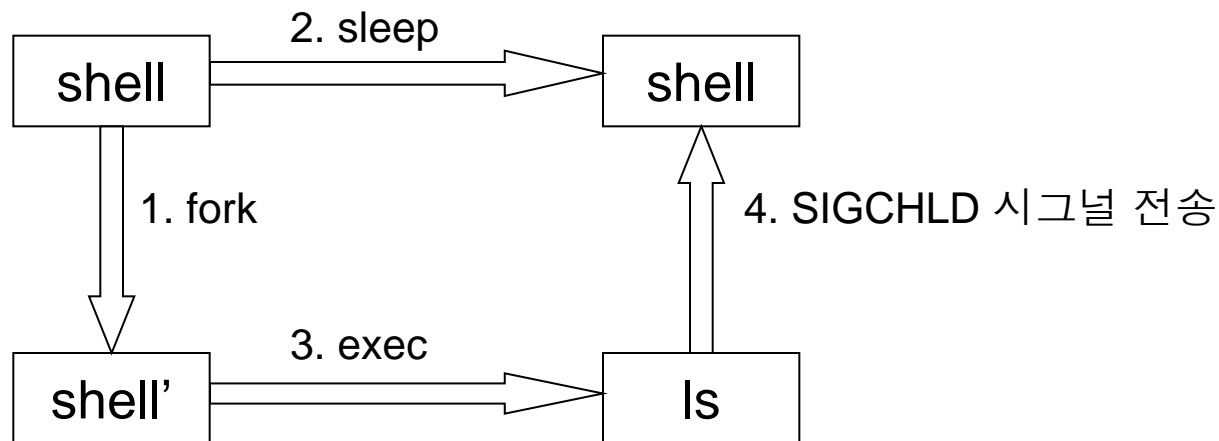
- 예)

\$ mkdir jhryu; cd jhryu

- 명령이 동시에 수행되는 것이 아닌 순차적 수행의 개념

셸의 명령어 수행 과정

■ “ls -l” 명령어 수행 과정





전개

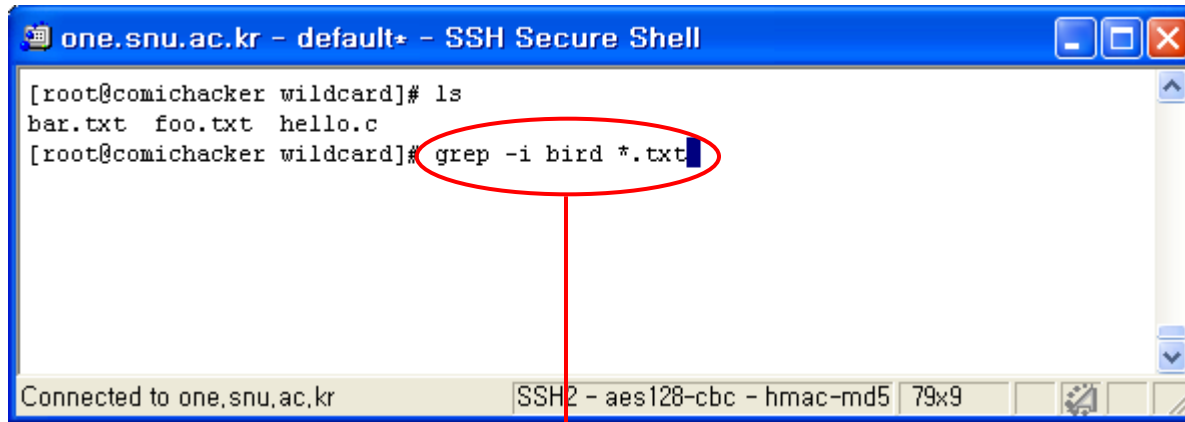
- 와일드카드 전개
- 변수 전개
- 명령 전개 (` `)
- 인용부호 제어 (‘ ‘ 또는 “ “)

와일드 카드 전개

- ? - 한문자, * - 여러문자,
file.*t.gz
he??o.txt
- 한글자의 캐릭터
[a-c]*.txt
hello.[coh]
- 문자열 집합
{ha,pe,mee}t

와일드 카드 전개

- 파일명은 명령행에 셸이 직접 전개함



```
one.snu.ac.kr - default* - SSH Secure Shell
[root@comichacker wildcard]# ls
bar.txt  foo.txt  hello.c
[root@comichacker wildcard]# grep -i bird *.txt
```

셸은 “grep -i bird bar.txt foo.txt” 로 exec

인용 부호 및 명령 전개

■ 인용부호 사용

- 쉘에서 의미를 가지는 메타문자의 효과를 정지
- ‘ ‘ : 전개 금지
`$ echo '$PATH'`
- “ ” : 변수, 명령전개만 허용
`$ echo "$PATH"`
- ` ` : 명령전개
`$ ls -l `which ls``
- 인용부호나 메타문자를 직접 표시하고 싶다면 \를 붙일 것
– \는 \\로 표시

셸 변수와 환경 변수

■ 변수의 두 가지 종류

● 지역변수 (셸 변수)

- fork 시 자식 프로세스에게 변수 값이 전달되지 않음
- 선언방법

\$ varname = string

● 전역변수 (환경 변수)

- Fork 시 자식 프로세스에게 변수 값이 전달 됨
- 선언방법

\$ export varname = string

또는

\$ varname = string

\$ export varname

셸 변수와 환경 변수

■ 지역 변수 (셸 변수)

- 기본적으로 프로세스의 환경변수와 관련 없음
- (이름, 값)의 데이터베이스
- 셸 프로그래밍이나 셸 제어에 사용
- 보통 소문자 이름

`$ a=123`

`$ x=100`

`$ echo $x`

셸 변수와 환경 변수

■ 환경 변수

- 유닉스 프로세스의 속성 중 하나
- (이름, 값)의 데이터베이스
- 셸에서 제어 가능
- 보통 대문자 이름
- 중요 환경변수
 - HOME
 - PATH
 - TERM
 - MAIL
 - SHELL
 - USER
 - MANPATH

bash shell

■ bash shell의 활용

- 대화식 실행 : UNIX 명령의 실행
- 맞춤 환경 : 작업 환경 구축
 - .bash_profile
 - .bashrc
 - .bash_logout
- 프로그래밍 : 새로운 도구의 개발
 - 명령어 조합이 자유로움
 - 변수 정의 기능
 - 비교 판단, 제어 구조 포함

bash shell (2)

■ bash shell의 기능

- 히스토리(history) 기능
- 앨리어스(alias) 기능
- 작업 제어 기능

■ bash shell의 초기화

- \$HOME/.bashrc, \$HOME/.bash_profile
- \$HOME/.bash_logout, \$HOME/.bash_history

환경 변수

- Environment variables
- 사용자 환경을 지정하는 변수
- 셸에서 수행되는 프로그램들에게 환경 변수의 값을 전달(**exports**)
- 환경 변수의 정의
 - export VARNAME=string
- 환경 변수
 - TERM
 - 터미널 타입을 정의
 - export TERM=vt100
 - HOME
 - 홈 디렉토리의 절대 경로이름

환경 변수

- PATH
 - 쉘 변수 path와 동일한 내용 저장(다른 형식)
- USER
 - 사용자의 로그인 이름
- SHELL
 - 사용자 쉘의 절대 경로
- MAIL
 - 메일 파일의 경로 이름

환경 변수

■ PATH

- 디렉토리 이름들을 포함
- 명령어를 찾는데 사용
- e.g. :

\$ export PATH = \$PATH:.

■ HOME

- 홈 디렉토리의 절대 경로이름(absolute pathname)
- 인자를 사용하지 않거나 ~ 를 사용하여 cd 명령을 수행하면 C 셸은 \$HOME을 사용

■ SHELL

- subshell을 생성하는 스크립트나 명령을 수행하면 이 변수에 지정된 파일을 수행

umask

- 새 파일 및 디렉토리에 대한 허가 모드 지정
- 형식 : `umask [nnn]`
 - [nnn] : `chmod`와 마찬가지로 8진 코드
- `umask`로 지정한 값의 보수값 사용
- 기본값(default)은 0022
 - 파일 생성시 644 (rw-r--r--)의 퍼미션을 가짐
 - 디렉 생성시 755 (rwxr-xr-x)의 퍼미션을 가짐
- 예)

```
$ umask 26
```

```
$ mkdir testdir
```

```
$ ls -ld testdir
```

```
drwxr-x--x 1 jhryu jhryu 1024 Apr 29 10:02 testdir/
```

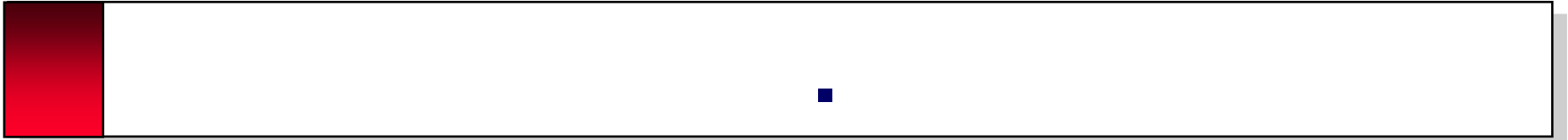


exit

- 쉘을 종료 시킴
- 새로운 쉘을 생성한 후 **exit**를 수행하면 부모 쉘로 돌아감

source

- 현재의 쉘에서 스크립트를 수행
 - fork 후 subshell에서 수행하는 것이 아님
- 현재의 환경을 바꾸는 데 주로 사용
- `$ source <스크립트 파일>`
- 예)
 - `$ source ~/.bashrc` : `.bashrc`의 모든 명령어를 수행

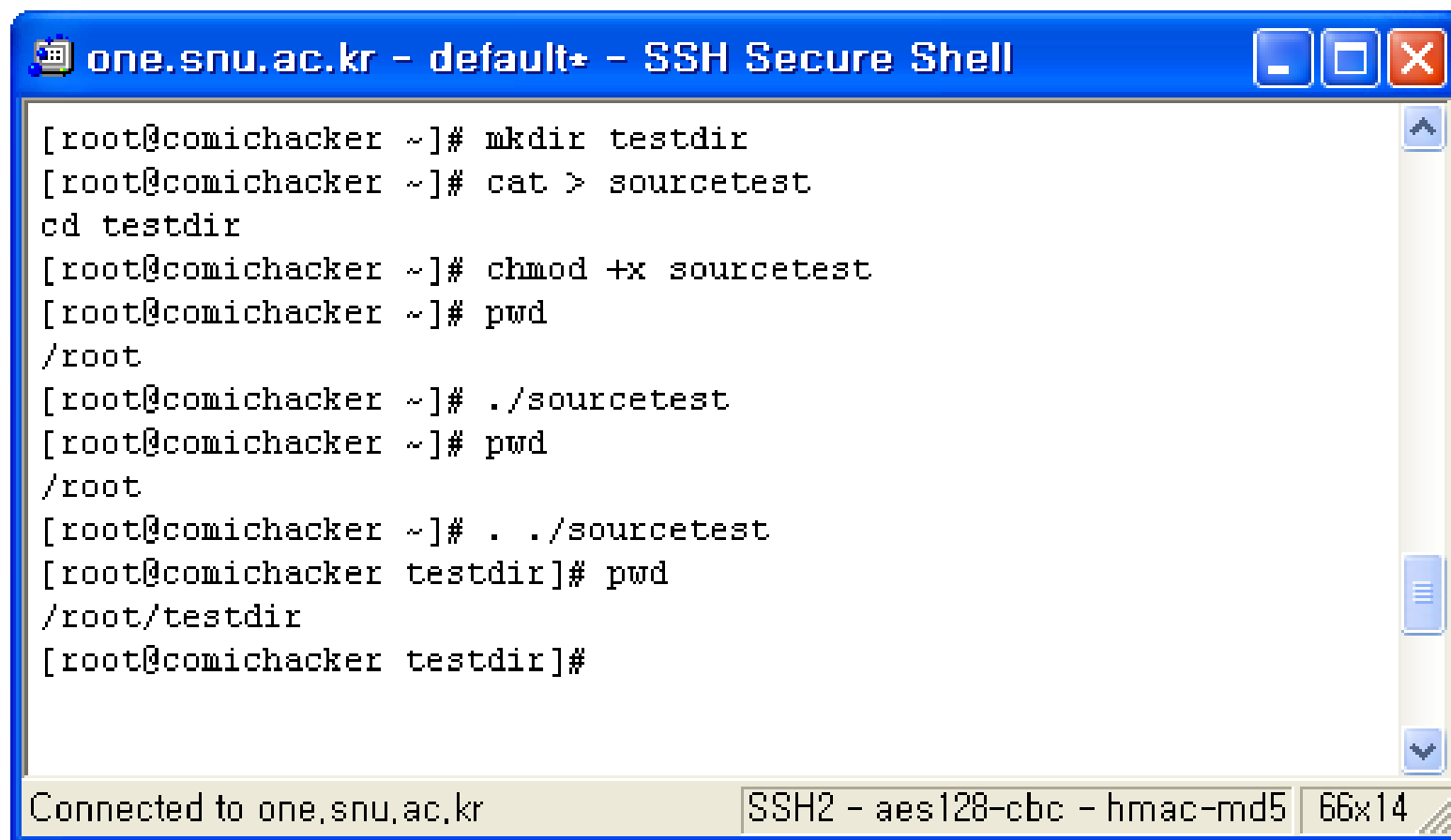


- source와 같은 기능

- 예)

- `$. ~/.bashrc` : `.bashrc`의 모든 명령어를 수행

source, .



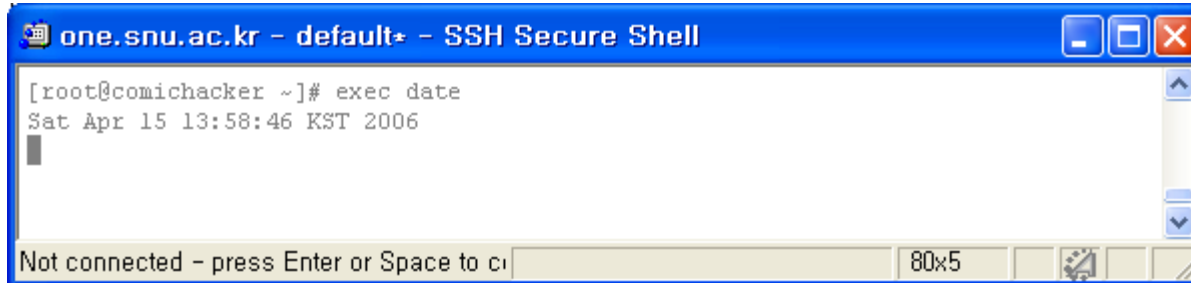
A screenshot of an SSH terminal window titled "one.snu.ac.kr - default* - SSH Secure Shell". The window shows a series of commands and their outputs. The user creates a directory named "testdir", creates a file named "sourcetest" with a cat command, changes its permissions to executable, and then runs it twice. The first run shows the current directory as "/root". The second run, preceded by a dot ".", shows the current directory as "/root/testdir". The terminal window has a status bar at the bottom showing "Connected to one.snu.ac.kr", "SSH2 - aes128-cbc - hmac-md5", and "66x14".

```
[root@comichacker ~]# mkdir testdir
[root@comichacker ~]# cat > sourcetest
cd testdir
[root@comichacker ~]# chmod +x sourcetest
[root@comichacker ~]# pwd
/root
[root@comichacker ~]# ./sourcetest
[root@comichacker ~]# pwd
/root
[root@comichacker ~]# . ./sourcetest
[root@comichacker testdir]# pwd
/root/testdir
[root@comichacker testdir]#
```

Connected to one.snu.ac.kr SSH2 - aes128-cbc - hmac-md5 66x14

exec

- 현재의 셸을 다른 프로세스로 대체
 - fork 하지 않고 **exec**을 바로 함
- **\$ exec <명령>**
- 예)



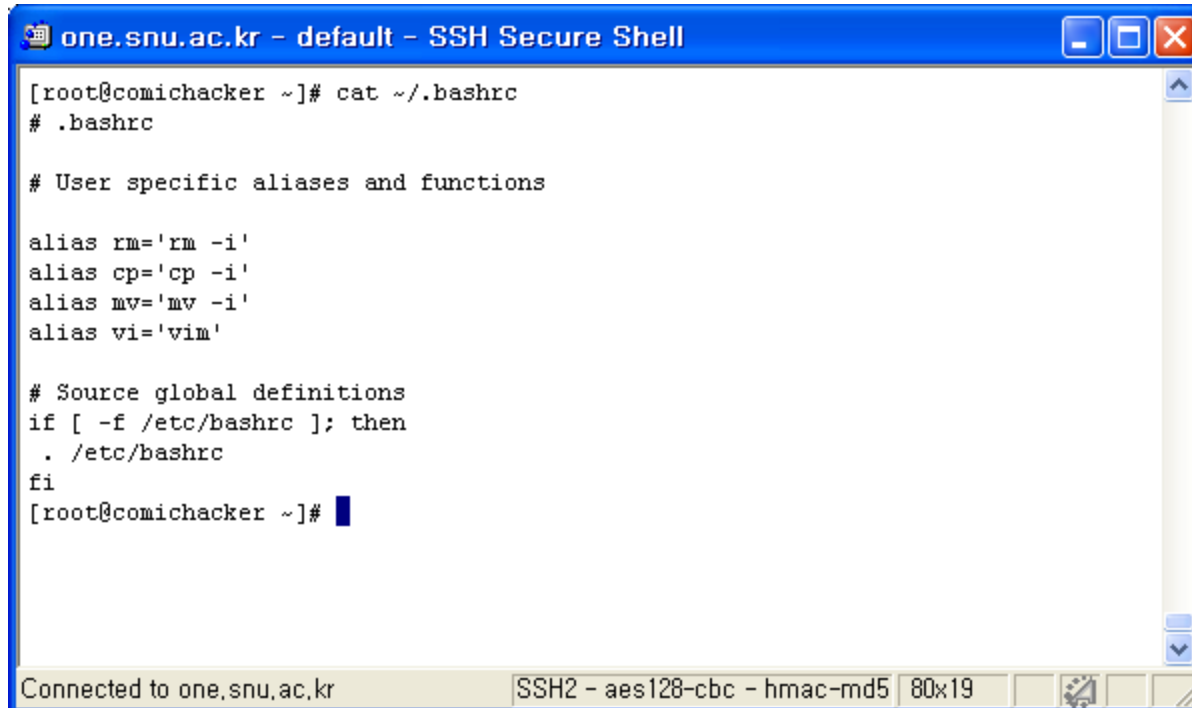
The screenshot shows a terminal window titled "one.snu.ac.kr - default* - SSH Secure Shell". The prompt is "[root@comichacker ~]#". The command "exec date" has been entered and executed, resulting in the output "Sat Apr 15 13:58:46 KST 2006". The status bar at the bottom indicates "Not connected - press Enter or Space to c" and "80x5".

```
[root@comichacker ~]# exec date
Sat Apr 15 13:58:46 KST 2006
```

.bashrc

- 로그인한 후 수행하는 스크립트
- `prompt`를 표시하기 전에 홈 디렉토리의 `.bashrc` 파일을 찾아 그 안의 명령들을 수행
- `.bashrc`는 쉘 변수와 `alias` 들을 정의하여 로그인할 때 자동적으로 설정되도록 하는데 주로 이용
- 쉘 스크립트를 수행할 때마다 `.bashrc`를 수행하므로 그 스크립트에게 `alias` 등의 설정이 사용 가능

.bashrc



The screenshot shows a terminal window titled "one.snu.ac.kr - default - SSH Secure Shell". The terminal displays the command `cat ~/.bashrc` and its output, which is the content of the `.bashrc` file. The output includes user-specific aliases and functions, and a section for sourcing global definitions. The terminal status bar at the bottom indicates the connection details: "Connected to one.snu.ac.kr", the encryption method "SSH2 - aes128-cbc - hmac-md5", and the window size "80x19".

```
[root@comichacker ~]# cat ~/.bashrc
# .bashrc

# User specific aliases and functions

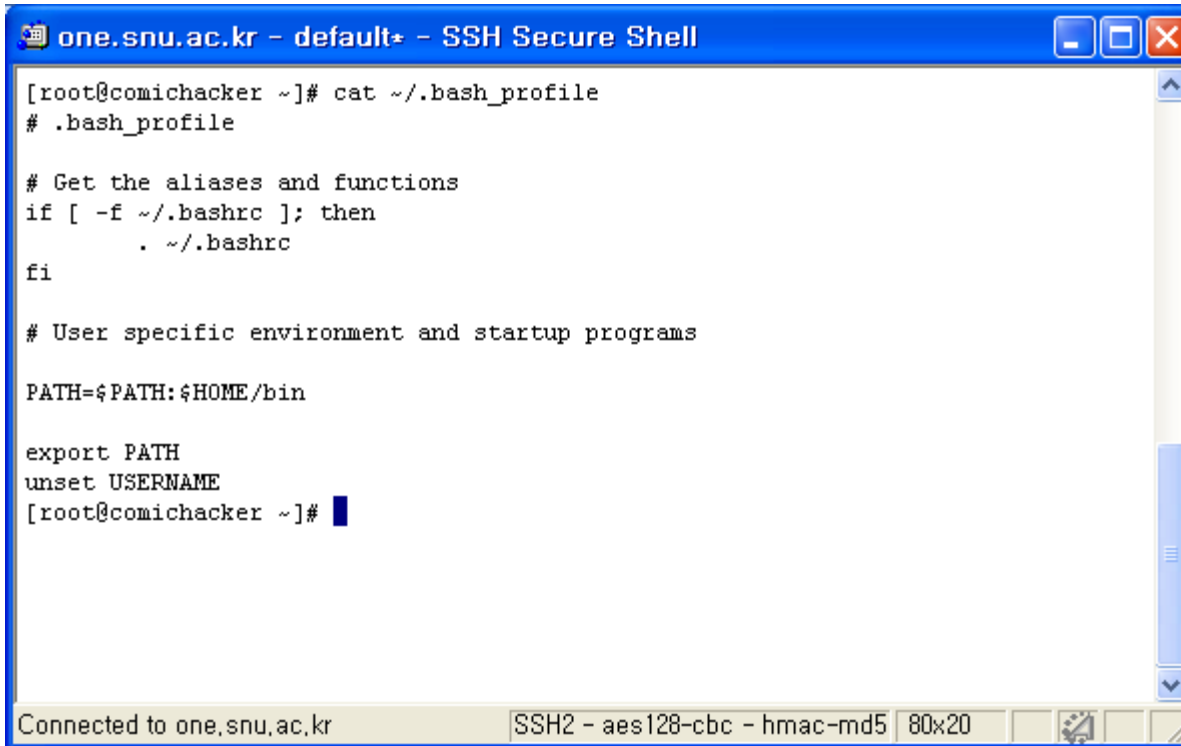
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias vi='vim'

# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
[root@comichacker ~]#
```

Connected to one.snu.ac.kr | SSH2 - aes128-cbc - hmac-md5 | 80x19

.bash_profile

- 각 세션마다 단 한번만 수행



The screenshot shows a terminal window titled "one.snu.ac.kr - default* - SSH Secure Shell". The terminal content is as follows:

```
[root@comichacker ~]# cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

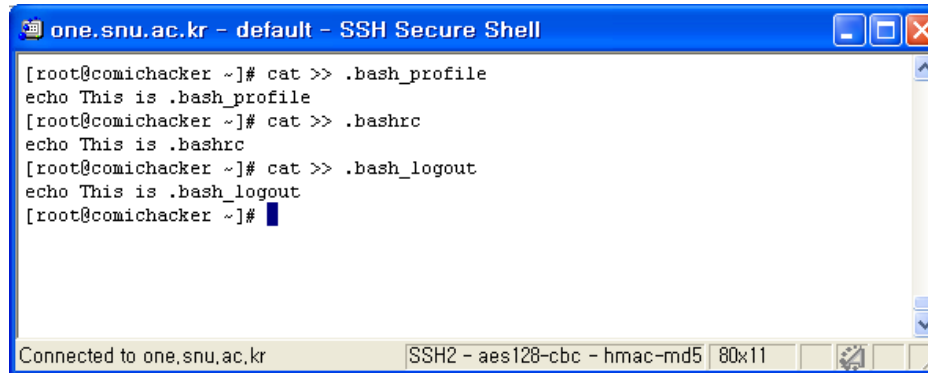
# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
unset USERNAME
[root@comichacker ~]#
```

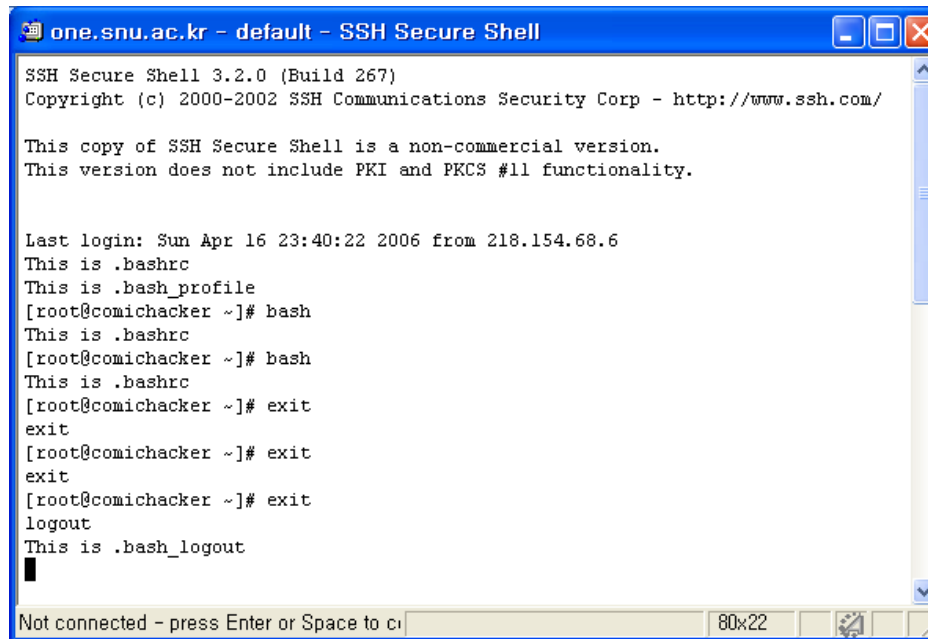
The terminal window also shows connection details at the bottom: "Connected to one.snu.ac.kr" and "SSH2 - aes128-cbc - hmac-md5 80x20".

.bash_profile



```
one.snu.ac.kr - default - SSH Secure Shell
[root@comichacker ~]# cat >> .bash_profile
echo This is .bash_profile
[root@comichacker ~]# cat >> .bashrc
echo This is .bashrc
[root@comichacker ~]# cat >> .bash_logout
echo This is .bash_logout
[root@comichacker ~]#
```

Connected to one.snu.ac.kr | SSH2 - aes128-cbc - hmac-md5 | 80x11



```
one.snu.ac.kr - default - SSH Secure Shell
SSH Secure Shell 3.2.0 (Build 267)
Copyright (c) 2000-2002 SSH Communications Security Corp - http://www.ssh.com/

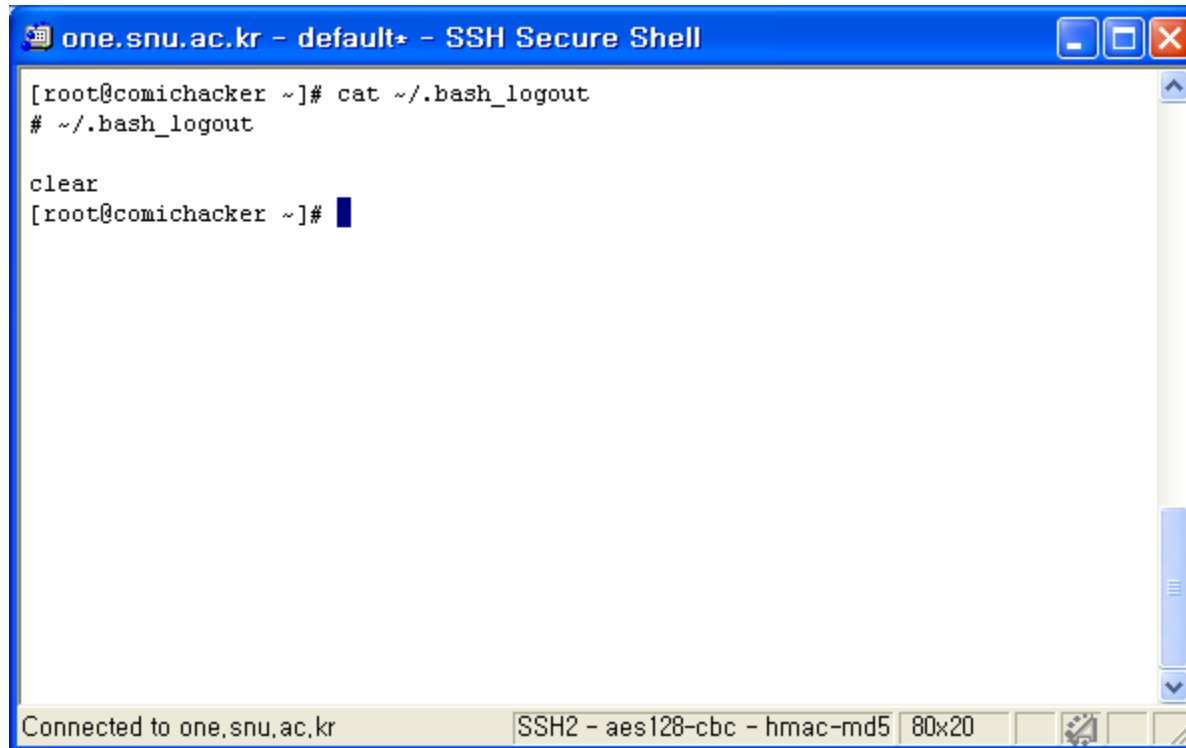
This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.

Last login: Sun Apr 16 23:40:22 2006 from 218.154.68.6
This is .bashrc
This is .bash_profile
[root@comichacker ~]# bash
This is .bashrc
[root@comichacker ~]# bash
This is .bashrc
[root@comichacker ~]# exit
exit
[root@comichacker ~]# exit
exit
[root@comichacker ~]# exit
logout
This is .bash_logout

```

Not connected - press Enter or Space to c | 80x22

.bash_logout



The screenshot shows a terminal window titled "one.snu.ac.kr - default* - SSH Secure Shell". The terminal content is as follows:

```
[root@comichacker ~]# cat ~/.bash_logout
# ~/.bash_logout

clear
[root@comichacker ~]#
```

The terminal window has a status bar at the bottom that reads: "Connected to one.snu.ac.kr", "SSH2 - aes128-cbc - hmac-md5", and "80x20".

bash shell의 기능

- 조건부 실행 (conditional execution)
- 명령 대치 (command substitution)
- 전위, 후위 프로세스 (foreground & background jobs)
- 히스토리 (history list)
- 별명 (aliases)

조건부 실행

■ <명령 1> && <명령 2>

- 명령 1이 성공하는 경우 명령 2 실행
- 앞 명령의 반환 값 (0: 성공, 0 이외의 값: 실패) 이용

■ 예)

```
$ grep text testfile > output && cat output
text
$
```

■ <명령 1> || <명령 2>

- 명령 1이 실패하면 명령 2 실행

■ 예)

```
% gcc test.c 2> err || mail root < err
% mail
Mail version 5.5 6/1/90. Type ? for help.
"/var/spool/mail/root": 4 messages 2 new 4 unread
.....
N 4 root@one.snu.ac. Mon Feb 12 11:20 11/421
```

~ and { }

■ 예)

```
$ cd ~jhryu  
$ pwd  
/user/jhryu  
$
```

■ 예)

```
$ cat acc.{jan,feb}  
.....  
$  
$ cp /usr/include/{stdio.h,math.h} .  
$
```

명령어의 대체

- 문자열 내에 명령어 결과의 삽입

```
% echo "there are `who | wc -l` users logged on"  
there are 8 users logged on  
%
```

\ 문자

- 두 라인 이상의 명령을 사용하려면?

```
% cp master.text /u/sam/utils/doc;
```

```
% cp master.text \  
/u/bill/backup
```

- \$ 자체를 출력하려면?

```
$ set name = jhryu
```

```
$ echo \ $name is $name.
```

```
$name is jhryu.
```

- * 자체를 출력하려면?

```
% echo \*
```

```
*
```

전위, 후위 프로세스

■ foreground 수행

- 명령어의 실행이 끝날 때까지 shell이 프롬프트를 출력하지 않고 기다림

■ background 수행

- 명령어가 끝나기를 기다릴 필요가 없이 다음 작업을 수행
- 명령어 끝에 ‘&’

■ 예)

```
$ tail -f test.txt &
$ ps
  PID TTY          TIME CMD
 7949 pts/25    0:01 csh
12481 pts/25    0:00 tail
$ kill 12481
$ kill -9 12481
```

프로세스 관련 명령

■ ps

- 현재 수행중인 프로세스의 목록을 보여줌

■ jobs

- 현재의 jobs들을 보여줌
- job : background로 수행 중이거나 stop된 프로세스

■ bg, fg

- 특정 프로세스를 background 나 foreground로 실행 상태를 변경

■ kill

- 특정 프로세스를 중지함

■ nohup

- logout후에도 명령을 계속 수행함

작업 제어 (job control)

■ 후위 작업

```
% (date; gcc -c test.c) 2> compile_log &
```

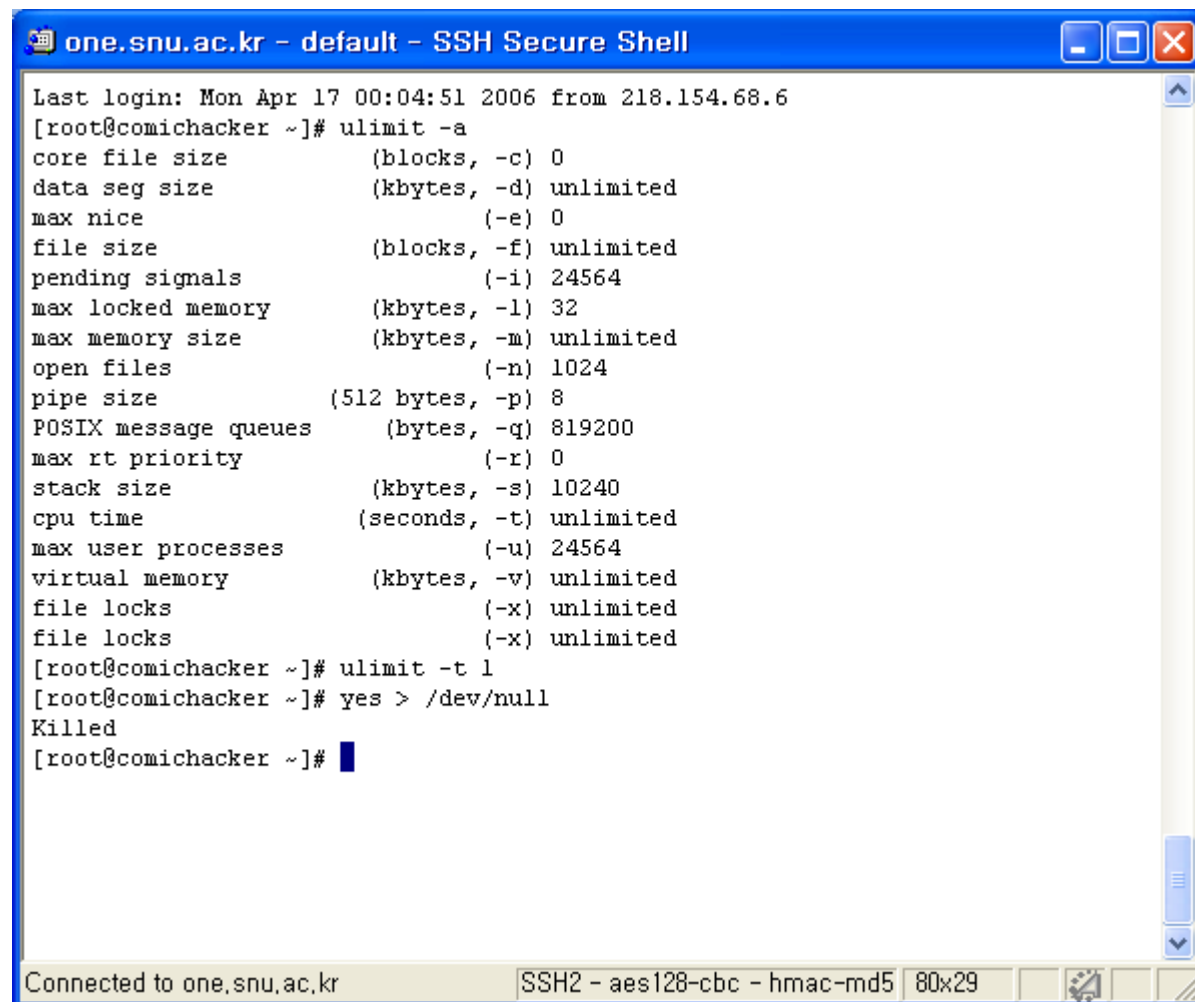
```
[1] 27412
```

```
%
```

```
[1] Done ( date; gcc -c test.c ) >& compile_log
```

```
%
```


CPU limit의 지정



A screenshot of an SSH terminal window titled "one.snu.ac.kr - default - SSH Secure Shell". The terminal shows the output of the command `ulimit -a`, which lists various system limits. The output is as follows:

```
Last login: Mon Apr 17 00:04:51 2006 from 218.154.68.6
[root@comichacker ~]# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
max nice                (-e) 0
file size               (blocks, -f) unlimited
pending signals         (-i) 24564
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
max rt priority         (-r) 0
stack size              (kbytes, -s) 10240
cpu time                (seconds, -t) unlimited
max user processes      (-u) 24564
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
file locks              (-x) unlimited
```

Following the `ulimit -a` command, the user enters `ulimit -t 1` to set a 1-second CPU time limit. Then, they enter `yes > /dev/null` to trigger a process that runs indefinitely. The terminal immediately displays "Killed", indicating that the process was terminated due to the CPU time limit.

```
[root@comichacker ~]# ulimit -t 1
[root@comichacker ~]# yes > /dev/null
Killed
[root@comichacker ~]#
```

The bottom status bar of the terminal window shows "Connected to one.snu.ac.kr", "SSH2 - aes128-cbc - hmac-md5", and "80x29".

작업

■ 백그라운드 프로세스의 상태를 출력

% jobs

[1] + Suspended (signal) yes > /dev/null

[2] - Running yes Hello > /dev/null

%

프로세스를 종료시키기

■ kill 명령

\$ kill %

\$ kill 26435

\$ kill %2

\$ kill %du

■ bash shell은 백그라운드 job 상태의 변화를 알려 줌

\$ kill %2

[2] Terminated

yes Hello > /dev/null

\$

작업 제어

■ 예)

```
% fg    %2
%
% stop  %3
% stop  %
%
% bg %3
% bg
%
```

후위 작업에 대한 입력

- 백그라운드 job이 입력이 필요하면 쉘은 작업을 멈추고 알려줌

```
$ rm -i ps_stat &  
[1] 27770  
$  
rm: remove `ps_stat'?  
[1] + Suspended (tty input) rm -i ps_stat  
$
```

- 정지된 job에 대해 입력을 주기 위해서는 전위 작업으로 변경하여야 함

```
$ fg %  
rm -i ps_stat  
y  
$
```

후위 작업에 대한 출력

- 후위 작업의 터미널 출력 막기

```
$ stty tostop
```

```
$ ls &
```

```
[1] 27941
```

```
$
```

```
[1]+ stopped
```

```
ls --tty=color
```

```
$
```

- 중단된 **job**을 전위로 바꾸면 정지된 출력을 계속함

```
$ fg %
```

```
ls
```

```
a.out
```

```
acc.jan
```

```
compile_log
```

```
test.c
```

```
acc.feb
```

```
acc.mar
```

```
newf2
```

```
test.o
```

```
$
```

히스토리

■ history list 의 출력

% history

1 ls -l

2 cat p.tst

3 cd UNIX-lect

4 ls

5 pa

6 ps

7 history

%

수행했던 명령어를 다시 수행하기

■ 바로 전 명령의 수행

- !!

■ 예)

```
$ !4
```

```
ls
```

```
total 22
```

```
1 ./      1 acc.feb      1 compile_log    1 test.o
```

```
1 ../     1 acc.jan      1 newf2
```

```
13 a.out*    1 acc.mar      1 test.c
```

```
$
```

```
$ !vi
```

```
$ ls
```

```
$ !!
```


지난 명령어를 수정하기

■ *^old^new*

```
$ lss -l acc.feb
```

```
lss: Command not found.
```

```
$ ^lss^ls
```

```
ls -l acc.feb
```

```
-rw-r--r--  1 jhryu  users      53 Feb 12 16:55 acc.feb
```

```
$
```

alias

■ **alias** 이름=정의

■ 예)

```
% alias h=history
% alias cx='chmod +x'
% cx acc.*
% alias
cls    clear
cx     (chmod +x)
%
% alias cx
chmod +x
%
% unalias cx
% cx acc.*
cx: Command not found.
%
```

alias의 중첩 사용

- alias 는 다른 alias를 사용할 수 있음

```
% alias h=history
```

```
% alias ah='h | head -3'
```

```
% ah
```

```
26 18:47 rm test.*
```

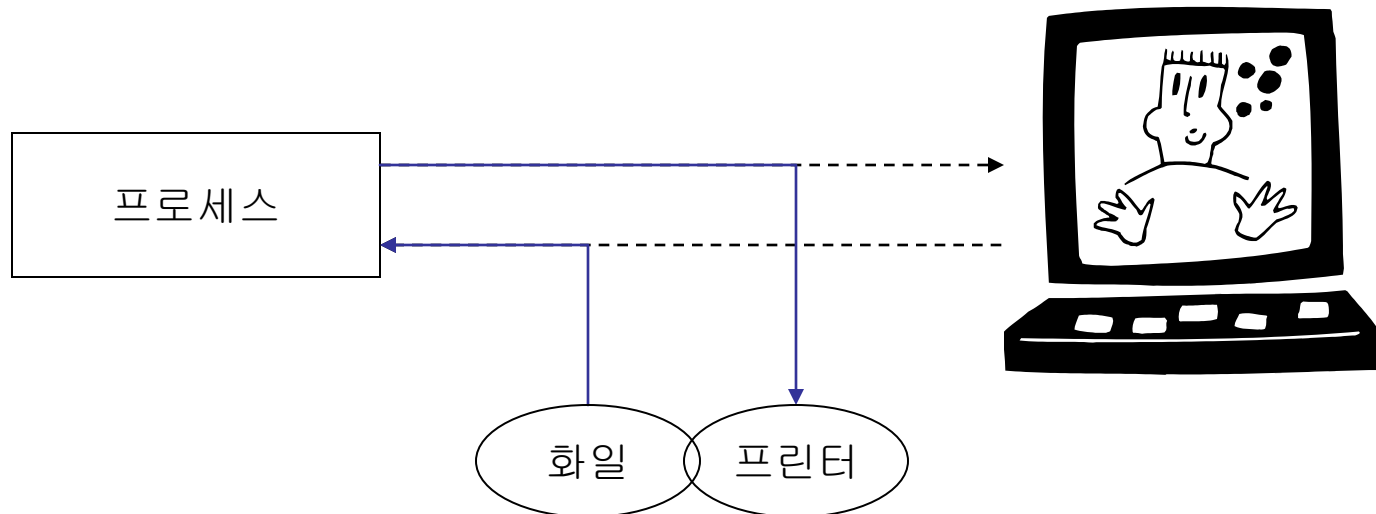
```
27 19:01 ls -l
```

```
28 19:03 ls *
```

```
%
```

표준입력, 표준 출력

- Unix에서는 터미널, 프린터, 디스크 등의 디바이스를 각각 하나의 화일로 취급
- 디바이스 파일
 - 예: /dev/tty06
- 기본적으로 표준 입출력 장치는 터미널



표준 출력 재지정

- 출력 재지정

- 명령 > file

- 예)

```
$ (date; du) > status
```

```
$ cat status
```

```
Mon Apr 28 20:47:27 KST 2006
```

```
1      ./test1
```

```
1      ./test2
```

```
10     .
```

```
$
```

출력 추가 (append)

- 명령 >> file

- 예)

```
$ cat status
```

```
Mon Apr 28 20:47:27 KST 2006
```

```
1      ./tmp
```

```
1      ./propose
```

```
10     .
```

```
$ ls -l >> status
```

```
$ cat status
```

```
Mon Apr 28 20:47:27 KST 2006
```

```
1      ./tmp
```

```
1      ./propose
```

```
10     .
```

```
total 9
```

```
-rw-r--r-- 1 jhryu  users  84 Jan 28 16:41 newf
```

표준 에러 출력 재지정

- 명령 2> file
- 명령 2>> file
- 예)

```
$ cc get.c 2> compile_out
```

```
$ cat compile_out
```

```
get.c: In function `main':
```

```
get.c:7: `wrong' undeclared (first use this function)
```

```
get.c:7: (Each undeclared identifier is reported only once
```

```
get.c:7: for each function it appears in.)
```

```
$ date; cc get.c -o get) >>& compile_log
```

표준 출력과 표준 에러의 분리

■ 예)

```
% (cat calendar logfile > save) >& errfile
```

```
% cat save
```

```
% cat errfile
```

```
cat: calendar: No such file or directory
```

```
cat: logfile: No such file or directory
```


안전한 출력 재지정

- set noclobber

- 예)

% set noclobber

% who > stat1

stat1: File exists.

% date >> date.log

date.log: No such file or directory.

표준 입력 재지정

- 명령 < file

- 예)

```
% tr "[A-Z]" "[a-z]" < oldf > oldf.low
```

```
% cat oldf.low
```

```
maryann clark 101
```

```
sally smith 113
```

```
jane bailey 121
```

```
jack austen 120
```

Pipes

- **Usage** : `command1 | command2 ... | commandn`

- **e.g.**

```
% who | wc -l
5
```

- **Usage:** `command1 | & command2`

필터

- 입력 데이터 스트림을 처리하여 출력 데이터 스트림을 만드는 명령어

- 예) sort

```
% ls | sort | tail  
temp  
test.ps  
test.txt  
tmp  
trace  
txttar.gz  
typescript  
util  
util.c  
zmodem
```

tee

- 하나의 입력으로 두 가지의 출력을 동시에 생성
 - 예)

```
$ who | tee who.out | grep jhryu
```

```
jhryu      pts/25    Aug 19 15:03  (one.snu.ac.kr)
```

```
$ cat who.out
```

```
jorc7911   pts/11    Aug 19 20:44  (comp38.snu.ac.kr)
```

```
limcom     pts/5     Aug 12 16:12  (beast)
```

```
jhryu      pts/25    Aug 19 15:03  (one.snu.ac.kr)
```



- vi 에디터 -

vi (1)

- BSD에서 개발
- 화면 편집기
- 터미널 제어 기능에 의존
 - System V : terminfo
 - BSD : termcap
- 터미널 설정

```
$ TERM = vt100; export TERM
% setenv TERM vt100
```
- 유닉스 사용에 필수적으로 알아야 함

vi (2)

■ vi 에디터의 실행과 종료

- 실행

\$ vi [filename]

- 종료

ZZ

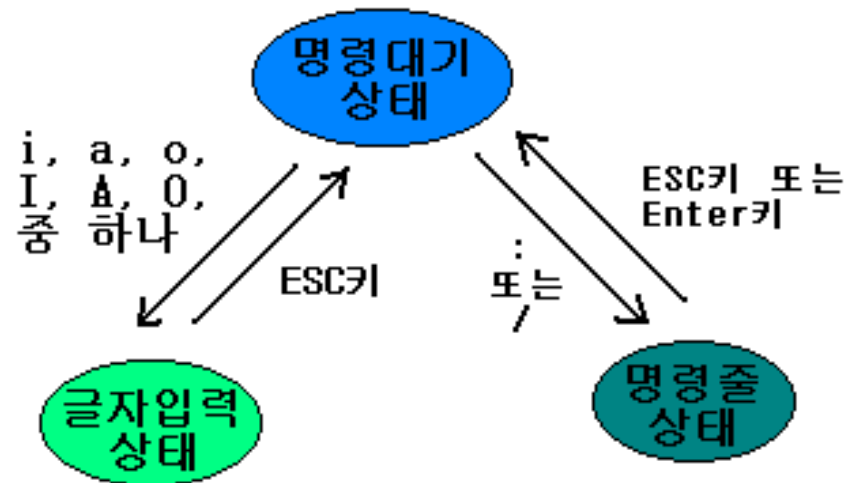
:wq

:X

■ 에디터의 사용 모드

- command mode
- input mode

■ 마지막 행 모드



vi (3)

■ 명령 모드

- 화면에서 입력하는 글자가 표시되지 않음
- 명령에 반응하여 실행
- 언제라도 **ESC**를 눌러 돌아갈 수 있음

■ 입력 모드

- 명령 모드에서 입력 명령 중 하나를 실행하여 전환
- 입력하는 문자가 화면에 표시

■ 마지막 행 모드

- 명령 모드에서 :를 입력하며 전환
- 화면 가장 아래에 : 프롬프트가 나타남
- 파일 관리나 검색/치환명령이 가능

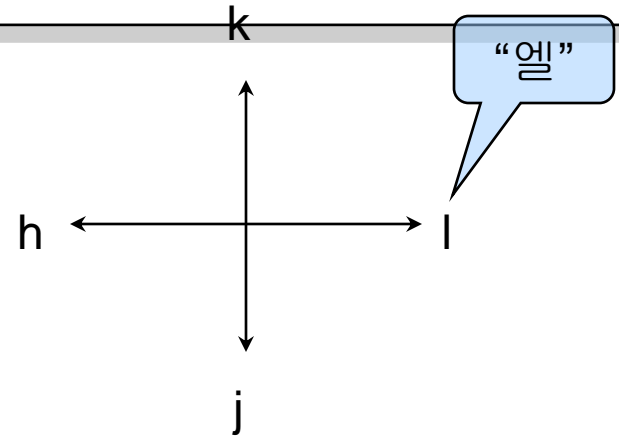
vi (4)

■ 명령 모드에서 커서 움직이기

- l : 오른쪽 한 칸 이동
- h : 왼쪽 한 칸 이동
- j : 아래로 한 라인 이동
- k : 위로 한 라인 이동
- w : 오른쪽 단어의 맨 앞으로 이동
- b : 단어의 맨 앞 또는 왼쪽 단어의 맨 앞으로 이동
- e : 단어의 맨 뒤 또는 오른쪽 단어의 맨 뒤로 이동
- W, B, E : w,b,e와 같지만 공백으로 단어 구분
- \$: 라인의 맨 끝으로 이동
- ^, 0 : 라인의 맨 처음으로 이동

■ 커서 이동 방식은 다른 유닉스 프로그램에서도 같이 사용

- more, less, elm, shell ...



vi (5)

- 명령 모드에서 커서 움직이기
 -) : 문자의 맨 끝으로 이동
 - (: 문장의 맨 처음으로 이동
 - } : 문단의 맨 끝으로 이동
 - { : 문단의 맨 처음으로 이동
 - H : 화면의 맨 위로 이동
 - M : 화면의 중간으로 이동
 - L : 화면의 맨 아래로 이동

vi (6)

- 명령 모드에서 커서 움직이기 (계속)
 - `]]` : 다음 함수로 이동
 - `[[` : 이전 함수로 이동
 - `^d` : 반 화면 아래로 이동
 - `^u` : 반 화면 위로 이동
 - `sf` : 한 화면 아래로 이동
 - `^b` : 한 화면 위로 이동
 - `nG` : n번째 라인으로 이동

vi (7)

- 명령 모드에서 입력 모드로 전환
 - i
 - I
 - a
 - A
 - o
 - O
 - r
 - R
- 삽입 모드에서 명령 모드로 바꾸기
 - ESC
 - ESC가 안되는 경우 - ^]

vi (8)

■ 실행 취소

- u

■ 지우기

- x
- X
- d [단위]
 - dd
 - d0(d^), d\$
 - dw, db, de, dW, dB, dE
 - d), d(
 - d}, d{
 - 5dd

vi (9)

■ 바꾸기

- c[단위]

- c0(c^), c\$
- cw, cb, ce, cW, cB, CE
- c), c(
- c}, c{

■ 교체

- r
- 3r

vi (10)

■ 문자열 찾기

- /문자열
- ?문자열
- n
- N

■ 문자열 찾아서 바꾸기

- :[범위] s/찾는 문자열/바꿀 문자열/[g]
- :s/windos/unix
- :1,\$s/windows/unix/g
- :.,+10s/windows/unix

vi (11)

■ 잘라내기, 복사하기 / 붙이기

- 잘라내기

- d
- 3dd

- 복사하기

- y
- 3yy

- 붙이기

- p
- P

■ 지시어 + 적용범위 = 명령어

- d(delete), y(yank), c(change)

vi (12)

- 파일의 읽기와 쓰기
 - :r filename
 - [범위]w[filename]
 - :w!
 - :wq!
- 다음 줄을 현재 줄 끝으로 밀어 올리기
 - J

vi (13)

■ vi에서의 매개 변수 설정

`:set [매개변수][=값]`

- 매개 변수
 - ai, noai : autoindent
 - cindent : c언어식의 indent
 - sm, nosm : show matching paranthesis
 - nu, nonu : 줄 번호 출력
 - showmode, noshowmode : 모드의 표시
 - tabstop [=값] : 탭의 크기
 - shiftwidth [=값] : indent의 크기
- \$HOME/.vimrc 에 저장

vi (14)

■ 키 매핑

- 약어 명령
 - :abbreviate(ab) tyu Teach Yourself UNIX in a Few Minutes
- 화살표 키를 각각 h, j, k, l 키로 매핑하기
 - :map ^v위화살표 k
 - 화면에는 ^[[A k 로 표시됨

■ .vimrc 파일

- 약어 명령이나 키 매핑 등을 미리 저장하여 vi 수행시마다 이용

■ !를 이용한 리눅스 명령

- :! → vi를 떠나지 않고 리눅스 셸(shell) 명령 수행
- !!명령어 → 명령어를 수행한 결과를 본문에 삽입

■ ^g : 파일의 현재 라인 번호 및 기타 정보 나열

vi (15)

- yank(문자열을 버퍼에 기억)
 - 10yy : 10줄을 버퍼에 기억
 - p(P) : 버퍼에 기억된 내용을 아래(위) 라인에 삽입
- 파일간 이동시에도 버퍼에 기억하기 위해서는?
 - “q10yy
 - q: 버퍼이름(임의의 문자)
 - “qp(P)
 - 버퍼 q에 저장된 내용을 꺼내 삽입
 - 예) abc 파일의 10라인을 cba 파일에 복사
 - vi abc -> 적절히 이동 -> “q10yy -> :r cba -> 적절히 이동 -> “qp -> :rew

vi (16)

■ 입력명령

- **s** : 현재 위치의 문자를 지우고 입력모드로 들어간다
- **S** : 현재 위치의 라인을 지우고 입력모드로 들어간다

■ 마크 기능

- 자주 참조하는 라인 및 문자에 **mark** 해 놓고, 나중에 찾아갈 때 사용
- **m[a-z]** : [a-z]의 이름으로 **mark**를 한다
- **'x** : mark 된 라인의 가장 앞으로 이동
- **`x** : mark 된 문자로 이동

■ Redo

- Undo (u)를 취소
- [ctrl+r]

vi (17)

■ 화면 분할

- **:[숫자]sp [파일이름]**
 - 현재 화면을 수평으로 [숫자]에 해당하는 라인으로 분할하며 [파일이름] 파일을 연다
- **:[숫자]vs [파일이름]**
 - sp와 다른 점은 수평 분할이 아니라 수직 분할을 한다는 점
- **[ctrl+w] + 방향키(h,j,k,l)**
 - 방향에 해당하는 윈도우로 커서를 이동

vi (18)

■ 네비게이션 바

- **:[숫자]vs [디렉토리이름]**

- 수직 분할 하되, 넓이는 [숫자] 크기이다. [디렉토리이름]에 들어 있는 파일들을 네비게이션 할 수 있게 된다.

- **[shift+o]**

- 열고 싶은 파일에 커서를 위치시키고, **[shift+o]** 키를 누르면, 해당 파일이 열린다. 편집 윈도우가 여러 개일 경우, 최근에 편집했던 창에 파일을 열게 된다.

■ Man page 참조

- **[shift+k]**

vi (19)

■ 블록 지정

- [ctrl+v]

- 블록 단위 블록 지정을 한다. 방향키(h,j,k,l)를 사용하여 지정

- [shift+v]

- 라인 단위 블록 지정을 한다. 상하 방향키(j,k)를 사용하여 지정

- [shift+v] [shift+g]

- 현재 라인부터 끝 라인까지 모두 블록 지정

■ 지정된 블록 처리 명령어

- y:복사, p:붙이기, d:삭제

- zf:접기, zo:펴기, zc:다시 접기

vi (20)

- 함수 및 변수명 자동 완성 기능
 - [ctrl+p]
 - 입력 모드에서 입력해야 함
- 쉬프트 명령
 - <<
 - 라인 또는 블록을 왼쪽으로 쉬프트
 - >>
 - 라인 또는 블록을 오른쪽으로 쉬프트
-]D,]d,[D,[d 명령
 - [D, [d
 - 매크로의 정의 부분을 화면 하단에 출력해 준다(커서 위로 검색)
 -]D,]d
 - 매크로의 정의 부분을 화면 하단에 출력해 준다(커서 아래로 검색)

vi (21)

■ 매크로 관련 명령

-] [ctrl+d
 - 매크로가 정의된 부분으로 이동(커서 위로 검색)
- [[ctrl+d
 - 매크로가 정의된 부분으로 이동(커서 아래로 검색)

■ 변수 선언 관련 명령

- gD, gd
 - 변수가 선언된 부분으로 커서를 이동

■ 브래킷 관련 명령

- %
 - 대응하는 브래킷 ('{,}' 또는 '[,']')을 찾아준다

vi (22)

■ 재 indent 기능

- =%

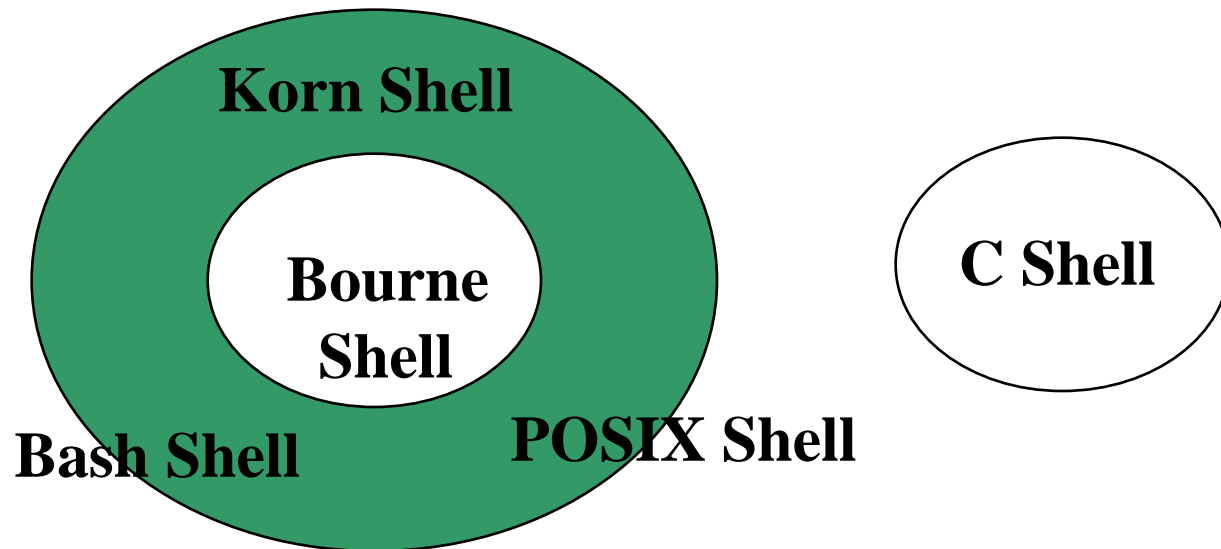
- 스쿼어 브래킷 ('{,}') 사이의 코드를 자동으로 indent 형식에 맞게 들여쓰기 해 줌



- 쉘 프로그래밍 -

셸

- 운영체제와 유저 프로그램 사이에 존재
- Bash 셸이 가장 많이 사용됨





bash

- 도구 프로그램 (utility)
- 명령어 해석기 (command interpreter)
 - 사용자 입력 명령어의 실행
- 프로그래밍 언어
 - shell script에 있는 명령을 실행

셸 스크립트의 작성 및 실행

- 셸 스크립트는 셸에 의해 수행될 명령어를 담은 파일
- 편집기로 셸 스크립트 작성

- \$ vi whoson

```
date  
echo Users Currently Logged In  
who
```

- 셸 스크립트에 대해 실행모드 허락
 - \$ chmod u+x whoson
- 스크립트의 이름을 호출하여 실행
 - \$ whoson 또는 \$./whoson, \$ sh whoson

명령어의 분리 및 묶음

■ <newline> 과 ;

- 세미콜론(;)을 사용함으로써 한 명령줄에 있는 일련의 명령어들을 분리할 수 있다. 그러나 <return>이 입력된 후에야 실행됨
- \$ a; b; c <return>

■ \

- 명령어가 한 줄을 넘어 다음줄에 연속될 때 사용

| 와 &

\$ a | b | c

\$ a & b & c

\$ a & b & c &

\$ a | b | c &

\$ (a ; b) & c

\$ (a ; b) & (c ; d) &

표준 에러의 재지정

```
$ cat x y 1> hold1 2> hold2
```

```
$ cat hold1
```

```
This is y
```

```
$ cat hold2
```

```
cat: cannot open x
```

```
$ cat x y 1> hold 2>&1
```

- 표준 출력은 hold, file descriptor 2는 file descriptor 1의 사본
- 즉, 표준출력과 표준에러는 hold

프로세스

■ fork

- 새로운 프로세스를 생성하는 운영체제 루틴의 이름
- 부모 프로세스는 자식 프로세스를 `fork(spawn)` 한다.

■ 프로세스의 구조

- 프로세스는 명령어의 수행으로서 파일시스템처럼 계층적 구조를 갖는다. `root process`는 사용자가 작업하는 모든 프로세스의 조상
- `getty` → `login` → `shell`

프로세스

■ 명령어의 실행

- parent process P forks a child process C
- P sleeps waiting to wake up
- C finishes execution and dies
- P wakes up and prompts for another command

■ 프로세스 ID

- 각 프로세스의 시작마다 고유의 프로세스 번호 소유
- 자식 프로세스는 부모 프로세스와 다른 번호 소유

\$ ps -l



변수 (variables)

- User-created variables
- Keyword shell variables
- Readonly shell variables



User-created variables

`$ person=alex` ; no space before and after =

`$ echo person` ; copy arguments to standard out
`person`

`$ echo $person` ; substitute the value of the variable
`alex`

`$ echo "$person"`
`alex`

User-created Variables

\$ echo '\$person' ; prevent variable substitution
\$person

\$ echo \ \$person
\$person

\$ person= ; 변수의 값을 삭제

\$ unset person

\$ readonly person ; 변수의 값 변경 불능

\$ readonly ; 사용자 정의 readonly 변수 나열

export 명령어

- export는 child 프로세스가 parent 프로세스의 변수를 접근할 수 있도록 하는 명령어 (일반적으로 변수는 그 변수가 선언된 프로세스에서만 접근 가능하다.)
- 파일 exctest1

```
cheese=american
echo "exctest1: $cheese"
subtest
echo "exctest1 2: $cheese"
```

export 명령어

■ 파일 subtest

```
echo "subtest 1: $cheese"
```

```
cheese=swiss
```

```
echo "subtest 2: $cheese"
```

```
% extest1
```

```
extest1 1: american
```

```
subtest 1:
```

```
subtest 2: swiss
```

```
extest1 2: american
```

export 명령어

■ 파일 exctest2

```
export cheese
```

```
cheese=american
```

```
echo "exctest2 1: $cheese"
```

```
subtest
```

```
echo "exctest2 2: $cheese"
```

```
$ exctest2
```

```
exctest2 1: american
```

```
subtest 1: american
```

```
subtest 2: swiss
```

```
exctest2 2: american
```

read 명령어

■ 파일 read1

```
echo -n "Go ahead: " ; suppress newline
read firstline
echo "You entered: $firstline"
```

\$ read1

Go ahead: This is a line.

You entered: This is a line.



명령어의 대처

\$ cat dir2

echo You are using the `pwd` directory.

\$ dir2

You are using the /home/jenny directory.

Keyword Shell Variables

HOME, PATH, MAIL, MAILPATH, MAILCHECK,
PS1, PS2, IFS, CDPATH, TZ

```
$ echo $HOME
```

```
$ PATH=/usr/ucb:/usr/bin:/usr/sbin:/home/jenny/bin:
```

```
$ export PATH
```

```
$ echo $MAIL
```

```
/var/mail/jenny (또는 /var/spool/mail/jenny)
```

```
$ PS1="`hostname` : "
```

```
bravo: echo test
```

```
test
```

```
bravo:
```

. command

■ .(dot) command

- runs the script as a part of the current process.

\$. .profile

\$. ./profile

- 다시 로그인 하지 않아도
- .profile의 변경내용이 즉시 효과가 있도록 함

Readonly Shell Variables

- \$0 호출한 프로그램 이름
- \$1, \$2, ... , \$9 인수
- \$* 모든 인수 지정 (“arg1 arg2 arg3...”)
- @\$ 모든 인수 지정 (“arg1” “arg2” ...)
- \$# 인수의 개수
- \$\$ 수행중인 쉘의 PID
- \$!
 - background에서 수행한 지난번 프로세스의 PID
- \$? 지난번 프로세스의 **exit** 상태
 - 0 → true, 성공적 수행
 - nonzero → false, 명령어 수행 실패

shift and set

■ shift

- promotes each of the command line arguments
- \$ shift ; arg1 ← arg2, arg2 ← arg3, ...

■ set

- command line arguments를 변수에 할당

```
$ cat set_it
set this is it
echo $3 $2 $1
$ set_it
it is this
```

흐름제어 명령어

- `if <test-com> then <coms> fi`
 `if [$# = 0]`
 `then`
 `echo Usage: chkargs argument, ... 1 >&2`
 `exit 1`
 `fi`
 `echo Program running`
 `exit 0`
- `if <test-com> then <coms> else <coms> fi`
- `if <test-com> then`
 `elif <test-com> then <coms> else <coms> fi`

흐름제어 명령어

- for <loop-index> in <arg-list> do <coms> done
for fruit in apples oranges pears bananas
do
 echo \$fruit
done
echo Task complete.
- break
 - done문 다음으로 이동하여 loop로부터 탈출
- continue
 - done문으로 이동하여 loop를 계속 수행

흐름제어 명령어

- while <test-com> do <coms> done

```
number=0
```

```
while [ "$number" -lt 10 ]
```

```
do
```

```
    echo "$number\c"
```

```
    number = `expr $number + 1`
```

```
done
```

```
echo
```

흐름제어 명령어

- `until <test-com> do <coms> done`
- `case <test-string> in`
 `pattern-1)`
 `coms-1`
 `::`
 `pattern-2)`
 `coms-2`
 `::`
`esac`

Here document

- 쉘 스크립트 자체로부터 입력을 받아들임

- 파일 birthday

```
grep -i "$1" <<+
```

```
Alex      June 22
```

```
Barbara   February 3
```

```
Jenny     January 23
```

```
Nancy     June 26
```

```
$ birthday Jenny
```

```
Jenny     January 23
```

exec

■ 셸에 내장된 프로그램

- 새로운 프로세스를 생성하지 않고 명령어를 실행

`exec date`

- 셸 스크립트 내에서 스크립트의 입출력을 재지정할 때 사용

`exec < infile`

`exec > outfile 2> errfile`

■ `exec`와 `.`의 차이점

- `.`은 스크립트만을 실행하나 `exec`은 스크립트와 컴파일된 프로그램을 모두 실행. 실행 후 `.`은 제어를 원래 스크립트에게 되돌려 주나, `exec`은 그렇지 않다. 새로운 프로세스를 생성하지 않으므로 `exec`은 실행속도가 빠르다.

trap

- 스크립트가 interrupt, illegal instruction, bad system call 등의 시그널을 받았을 때 취할 행동을 지정

1	hang up
2	terminal interrupt (DEL or ^C)
3	quit (^I or ^\)
9	kill (kill -9)
15	kill (software termination)
18	stop (^Z)

```
trap 'echo PROGRAM INTERRUPTED; exit 1' 2
```

```
trap 'exit 1' 1 2 3 15
```

```
trap 'rm /tmp/$$. $script 2> /dev/null' 0
```


셸 함수 (Shell Function)

- 셸 함수는 셸 스크립트처럼 일련의 명령어를 저장하지만, 파일이 아니라 주기억장치에 저장한다. 빨리 수행됨. 셸 함수는 호출한 셸과 같은 셸에서 수행됨

```
go()  
{  
    cd $1  
    PS1="[ `pwd` ] "  
}
```

```
$ pwd  
/home/alex  
$ go literature  
[ /home/alex/literature ]
```