

효율적인 리눅스 관리

입력과 출력 그리고 파이프 라인

man으로 명령어 설명 확인가능

stdin(표준 입력)

- 리눅스 운영체제가 키보드로부터 입력을 받는 입력스트림
- 프롬프트에서 명령을 입력하면 바로 이 표준 입력을 통해 명령이 입력됨
- stdout(표준 출력)
 - 결과를 출력하는 출력스트림
- "|"을 통한 명령어 결합
 - less /bin 과 ls -l /bin
 - less /bin은 stdout을 사용한다 -> outputbuffer를 사용한다는 의미 -> 즉 한글자씩 출력
 - ls -l /bin은 stdin을 사용 -> inputbuffer를 사용 -> 개행문제가 사용되면 모니터에 출력

wc를 이용한 집계

- wc는 파일 내용의 줄, 단어, 글자 수를 세어 출력해준다.
- l : line
- w : word
- c : char
- head/tail를 이용한 읽기
 - head
 - 위에서 부터 읽을때 사용
 - n으로 줄수 선택 가능 -> 생략했을때는 자동으로 -n10으로 됨
 - tail
 - 하단에서 부터 읽을때 사용
 - f로 실시간으로 읽기 가능

cut으로 특정 열만 읽기

- ex) cut -f filename.txt
- f : tab을 구분자로 사용해서 특정 열 읽기 가능
 - f1 = python / snail / robin /...
 - f2 = Programming Python / SSH /...

```
python Programming Python 2010 Lutz, Mark
snail SSH, The Secure Shell 2005 Barrett, Daniel
alpaca Intermediate Perl 2012 Schwartz, Randal
robin MySQL High Availability 2014 Bell, Charles
horse Linux in a Nutshell 2009 Siever, Ellen
donkey Cisco IOS in a Nutshell 2005 Boney, James
oryx Writing Word Macros 1999 Roman, Steven
~
~
```

c : 글자 수대로 출력 가능

- ex) cut -c1-5 filename.txt

grep으로 특정 문자열 찾기

- ex) grep [find text] [filename]
 - v를 사용하여 포함하지 않는 줄 찾기
 - ex) grep -v [find text] [file name]
 - 와일드 카드를 사용하요 모든 파일에서 찾기
 - ex) grep [fund text] .

```
~/linux_use/perface/ch02/command_4_grep % master
> grep Perl *.
animals.txt:alpaca Intermediate Perl 2012 Schwartz, Randal
essay.txt:really love the Perl programming language, which is
essay.txt:languages such as Perl, Python, PHP, and Ruby
~/linux_use/perface/ch02/command_4_grep %
```

- grep -w
 - 일치하는 단어만
- grep -i
 - 대소문자 무시
- grep -iw this [file pattern]
- grep -l his *
- 정규식을 이용한 grep 사용 가능
 - grep '^[A-Z]' [file pattern]

fgrep을 이용한 특정 문자열 찾기

- grep과는 다르게 문자 그대로 찾도록 하는 fixed기능이 있다.
- fgrep w. [file name]

sort로 정렬

- ex) sort [file name]
- -r로 내림차순 정렬
 - sort -r [file name]
- -h로 크기 단위 정렬
 - sort -rh [file name]
 - du -sh /bin/* | sort -rh (du 파일 용량 보기)
- -n으로 숫자 단위 정렬
 - sort -rn [file name]

uniq로 반복되는 내용 확인

- ex) uniq [file name]
- 근데 여기서 반복되는 문자라 함은 다음 과 같다
 - AAABBC 이렇게 정렬이 되어 있는 상태면 A3, B2 ,C1가 되고
 - ABCABA이렇게 정렬이 되지 않는 상태명 A1,B1,C1,A2,B2,A3가 됨
 - 즉, 정렬이 이루어 져야 의미가 있음
- -c를 통해 반복횟수 표시
 - ex) uniq -c [file name]

중복 파일 찾아내기

- 파일이 생성되면 고유문자열을 자동으로 생성한다. 해당 문자열은 md5sum으로 확인할 수 있다.

```
~/linux_use/perface/ch01/detecting_duplicate_files P master
md5sum ./*
146b163929b6533f02e91bdf21cb9563 ./image001.jpg
63da88b3ddde0843c94269638dfa6958 ./image002.jpg
146b163929b6533f02e91bdf21cb9563 ./image003.jpg
17f339ed03733f402f74cf386209aeb3 ./image004.jpg
381ebc2cd3aab91a65492ef360714e2c ./image005.jpg
1aa30608eb268a45266403f177f214d0 ./image006.jpg
f6464ed766daca87ba407aede21c8fcc ./image007.jpg
b965d5f3463e41eb66ea87a2933c407c ./image008.jpg
8f8d01a6598833fb04abc69fe9e9572c ./image009.jpg
bc64c99757e199ef858f52acf7e4e836 ./image010.jpg
714ecee06b43c03fe20eb96474f69b8 ./image011.jpg
f6464ed766daca87ba407aede21c8fcc ./image012.jpg
3825f1cffa61aee4673f5b7c535b2a09 ./image013.jpg
f6464ed766daca87ba407aede21c8fcc ./image014.jpg
1258012d57050ef6005739d0e6f6a257 ./image015.jpg
c96a1094226ad766fc9367f508dc9b32 ./image016.jpg
bef69f30a2f88a20e81797ea65c1e082 ./image017.jpg
d8ad913044a51408ec1ed8a204ea9502 ./image018.jpg
c7978522c58425f6af3f095ef1de1cd5 ./image019.jpg
c7978522c58425f6af3f095ef1de1cd5 ./image020.jpg
```

- 여기서 이제 중복이 몇개인지 확인하기 위해서는 정렬 → 반복내용 체크를 진행하면 된다.
- `md5sum * | cut -c1-32 | sort | uniq -c`

shell과 친해지기

와일드카드를 이용

grep [text] [file name]

- file name을 와일드 카드로 제한 가능하다
- * : 모든 내용 가능
 - 확장자 제한 가능 ex) `ls -al *.conf`
 - 파일명 제한 가능 ex) `ls -al filename.*`
- ? : 숫자만 올 수 있음
 - ex) `grep Linux chapter?`
 - ?의 수량에 따라서 숫자의 크기가 달라짐

```
> grep Linux chapter?
chapter1:This file contains the word Linux
chapter4:This file contains the word Linux
chapter7:This file contains the word Linux
```

```

▶ grep Linux chapter??
chapter10:This file contains the word Linux
chapter13:This file contains the word Linux
chapter16:This file contains the word Linux
chapter19:This file contains the word Linux
chapter22:This file contains the word Linux
chapter25:This file contains the word Linux
chapter28:This file contains the word Linux
chapter31:This file contains the word Linux
chapter34:This file contains the word Linux
chapter37:This file contains the word Linux
chapter40:This file contains the word Linux
chapter43:This file contains the word Linux
chapter46:This file contains the word Linux
chapter49:This file contains the word Linux
chapter52:This file contains the word Linux
chapter55:This file contains the word Linux
chapter58:This file contains the word Linux
chapter61:This file contains the word Linux
chapter64:This file contains the word Linux
chapter67:This file contains the word Linux
chapter70:This file contains the word Linux
chapter73:This file contains the word Linux
chapter76:This file contains the word Linux
chapter79:This file contains the word Linux
chapter82:This file contains the word Linux
chapter85:This file contains the word Linux
chapter88:This file contains the word Linux
chapter91:This file contains the word Linux
chapter94:This file contains the word Linux
chapter97:This file contains the word Linux

```

- [1-9]로 글자의 범위를 지정할 수 있음
 - ex) grep Linux chapter[1-5]

printenv로 환경변수 불러오기

- printenv HOME → /home/hwangjoonsoung
- printenv USER → hwangjoonsoung
- 변수 설정하기
 - [key]=[value]
 - ex) intel=%HOME/intellij → echo \$intel
 - 패턴과 변수를 사용하여 파일 한번에 옮기기
 - *를 이용하면 모든 파일을 읽을 수 있다.
 - ex) echo /bin/*
 - 이를 이용해서 파일을 한번에 옮길 수 있다
 - ex) mv /bin/*.txt \$HOME

alias를 이용한 명령어 단축하기

- alias를 통해 명령어를 단축시킬 수 있으며 쉽게 사용할 수 있다.
- ex) alias g=grep → g [text]
- alias를 통해 조회 또한 할 수 있다
 - alias

```

alias
-= 'cd -'
...=../..
....=../..
.....=../..
.....=../..
1='cd -1'
2='cd -2'
3='cd -3'
4='cd -4'
5='cd -5'
6='cd -6'
7='cd -7'
8='cd -8'
9='cd -9'
_='sudo '
egrep='grep -E'
fgrep='grep -F'
g=grep
ga='git add'
gaa='git add --all'
gam='git am'
gama='git am --abort'
gamc='git am --continue'
gams='git am --skip'
gamscp='git am --show-current-patch'
gap='git apply'
gapa='git add --patch'
gapt='git apply --3way'
gau='git add --update'
gav='git add --verbose'

```

- 보면 git과 같은 application에서 제공하는 것도 확인 할 수 있다
- 별명을 취소할 때는 unalias를 사용하면 됨
 - ex) unalias g

> >>을 이용하여 파일에 바로 쓰기

- grep을 사용하면 해당하는 text를 찾을 수 있다.
- 이때 >를 통해 file을 생성하고 동시에 파일에 text를 작성할 수 있다.
 - ex) grep [text] [file name] > [new file name]
 - alias | grep git > "git command"
 - cat git command
- 파일 마지막 줄에 추가로 text를 입력하고 싶은 경우 >>를 사용한다
 - ex) echo "test" >> "git command"
- 표준 오류 스트림을 입력할 때는 2> , 2>>를 사용
 - 리눅스 오류가 발생했을 때 표준 출력이 출력을 담당하는 것이 아닌 표준 오류가 출력을 담당함

때문에 표준 오류 스트림을 file에 입력할 때는 2>를 사용해야 한다.

- 표준 오류, 표준 출력 스트림 구별없이 사용할때는 &>를 사용한다

"와 '의 사용

- 명령어는 \s로 구분한다.
- 이를 하나의 단어로 인식하게 하기 위해서 ", '를 사용하는데 차이는 다음과 같다.

```
~/linux_use/perface/ch02/patterns_vs_variables/reptiles ♪ master ±
▶ echo 'this is home directory : $HOME'
this is home directory : $HOME
~/linux_use/perface/ch02/patterns_vs_variables/reptiles ♪ master ±
▶ echo "this is home directory : $HOME"
this is home directory : /Users/hwangjoonsoung
```

- 위와같이 문자로 받아드리게 할때는 '를, 이스케이프 와 같이 인식하게 하기 위해서는 "를 사용한다.

셸이 명령어를 찾는 방법

- 기본적으로 검색할 디렉토리 목록(\$PATH)이 있고 그 목록을 순회하면서 찾는다.
- try) echo \$PATH | tr "\n"

1. 명령어가 입력됨

2. \$PATH에 있는 경로를 한번 씩 순회하면서 해당 명령어가 있는지 찾아봄

which, type 를 통해 실행할 프로그램 찾기

- 내가 입력한 명령어가 어디 위치해 있는지 찾을수 있다.
- type의 경우 별명, 함수, 셸 내장 명령들을 찾는다.
- ex) which ls
- ex) type ls

정리

- 셸은 명령을 실행하기 전 입력받은 명령행을 평가한다
- 명령을 실행할 때 표준입력, 표준 출력, 표준 오류 스트림을 다른 곳으로 리다이렉트 할 수 있다
- ", '를 사용하면 특수 문자가 우너래 의미 대신문자 그대로 평가된다.
- 셸은 검색 경로에 포함된 디렉터리 모곡을 차례로 뒤지며 실행할 프로그램을 찾는다.
- \$HOME/.bashrc 파일의 내용을 통해셸의 기본 동작 설정을 바꿀 수 있다.

실행했던 명령을 다시 실행하기

history를 통해 입력했던 명령어 찾기

- ex)
 - history
 - history 3
 - history | less
 - history | sort -nr | less

명령 히스토리에서 이전 명령 불러오기

- history size
 - echo \$HISTSIZE
- !를 이용한 이전 명령어
 - !!를 사용하면 이전 명령어가 command line에 적용
 - ![command]를 하면 이전에 사용한 command를 찾아와서 command line에 적용
 - ex) !ls → ls -al
- 번호를 통한 command line 입력
 - history를 하면 왼편에 숫자가 나옴
 - !1202를 입력함으로써 해당 command 입력 가능
- 파일 삭제를 진행할때 실수 방지 방법
 - 파일명을 먼저 찾아보는 습관을 들이자
 - 1. ls를 통한 파일 확인 ex) ls [file name]
 - 2. !\$를 통해 가장 최근에 입력한 명령의 마지막 단어를 입력 rm -i !\$
 - 3. file name을 확인 하면서 1개씩 삭제
- ctrl+r을 이용한 검색
 - ctrl+r을 이용하면 명령을 검색할 수 있다.

명령어 변경

- ^를 이용한 명령어 변경

- `^[target]^[replace]`
- `md5sum *.jp | cut -1-32 | sort | uniq -c | sort -nr` 이런 명령어를 날리는데 jp가 아니라 jpg였다면?
- `^jp^jpg`를 이용하면 된다
- 자동으로 command line에 해당 명령이 변환되어 들어간다.

정리

- 삭제할 때는 ls를 통해서 파일을 한번 검색하고 `rm -i !$`를 통해 삭제하도록 하자
- 명령어 검색할 때 `ctrl+r`을 이용하면 간편하게 찾을 수 있다.
- no more history

파일 시스템을 자유롭게 이동하기

HOME을 쉽게 이동하는 방법

- ~를 이용한 이동
 - `cd ~`
 - `cd $HOME`
 - 다른 사용자로 이동
 - `~[other user]`

CDPATH를 이용한 폴더 이동

- 쉘이 명령어를 찾을 때 \$PATH를 이용한다.
- 이와 같은 이치로 \$CDPATH를 이용하면 빠르게 디렉토리를 이동할 수 있다.
- \$CDPATH에 디렉토리를 저장하면 저장된 디렉토리를 순회하면서 파일이나 폴더가 있는지 찾아본다
 - `CDPATH=$HOME:$HOME/intellij:..`
 - ..를 추가함으로 써 상위 디렉토리로 바로 이동 가능
 1. `ls ..`
 2. `cd [tab]`
- 이전 디렉토리로 이동
 - `cd -`

- 디렉터리 스택을 이용한 이동
 - 스택 구조로 이뤄져 있으며 "pushd ./"를 이용해 경로를 저장 popd를 이용해 해당 디렉터리로 이동한다.
 - 스택 추가
 - pushd [pwd]
 - 모든 스택 보기
 - dirs -p
 - dirs -v
 - 해당 스택으로 이동
 - popd
 - pushd -[N], pushd +[N] : 중간 스택에 있는 dir로 이동
 - pushd -[N] : 뒤에서 N번째
 - pushd +[N] : 앞에서 N번째
 - 스택의 변화
 - pushd +3을 했을때

Index	Path
0	~/Movies
1	/Volumes
2	~
3	~/intelliJ
4	~/Desktop

Index	Path
0	~/intelliJ
1	~/Desktop
2	~/Movies
3	/Volumes
4	~

- 스택 스왑
 - pushd
 - [dir1] [dir2] [dir3] → [dir2] [dir1] [dir3]
- 잘못 이동한 경우 (잘못이동해서 스택이 하나 pop된 경우)
 - 둘다 이전 디렉터리로 이동시켜주는건 변함이 없음
 - popd - 와 pushd -p의 차이
 - popd -[N] : 해당 스택 삭제

- `dirs` → ~ /Volumes ~/intelliJ
- `popd -1` ~ ~/intelliJ
- `pushd -` : 지워진 스택을 하나 살리고 해당 스택으로 이동
 - `dirs` → ~ /Volumes ~/intelliJ
 - `popd` → /Volumes ~/intelliJ
 - `pushd -` → ~ /Volumes ~/intelliJ
- 스택이 없는 경우
 - `directory stack empty` 안내가 나온다.

리눅스 명령을 몸에 익히기

텍스트 생성하기

- 날짜
 - `date`
 - `date +%Y-%m-%d` → 2025-09-26
 - `date +%H:%M:%S %A` → 23:23:59 목요일
- 시퀀스-연속 숫자
 - `seq`
 - `seq 1 5` → 1 2 3 4 5
 - `seq 1 2 10` → 1 3 5 7 9
 - `seq 10 -1 0` → 10 9 8 7 6 5 4 3 2 1 0
 - 구분자 추가
 - `seq -s[구분자] 0 2 10` → 0/2/4/6/8/10
 - 자리수 일치 (10이 두자리 수 이기때문에 0을 채움)
 - `seq -w 0 2 10` → 00 02 04 06 08 10
- 중괄호
 - `echo {1..10}`
 - `echo {10..1}`
 - `echo {00..10}`

- `echo {1..100..10}` → 1부터 100 까지 10씩 더해서
- `echp {001..1000..100}` → 1부터 1000까지 100씩 더해서 (자리수 맞춤)
- []와 {}의 차이
 - ||
 - `file1 file2 file3 file4`
 - 일치하는 파일만 찾는 케이스
 - `ll file[3-5]` → `file3 file4`
 - 순회를 하면서 파일이 있는지 찾는 케이스
 - `ll file{3..5}` → `file3 file4 file5 No such file or directory`
- find
 - `find [path] -print`
 - path 하위에 있는 모든 파일을 찾음
 - -print가 있어야 하는 이유는 posix표준 호환성(모든 리눅스 계열에서 사용가능한 명령)이기 때문에 필요
 - `find [path] -type [file type] -print`
 - path하위에 있는 모든 파올에소 file type에 해당하는 파일만 찾음
 - file type : f→파일 d→디렉터리
 - `file [path] -type f -name "[file pattern]" -print`
 - `file [path] -type f -iname "[file pattern]" -print`
 - ex) `find ./ type f -name "*.txt" -print`
 - file pattern에 해당 하는 파일만 찾음
 - 대소문자 구문하지 않기 위해서는 -iname을 사용
 - `find [path] -name [file pattern] -exec`
 - ex) `find ./type f -name "*.txt" -exec echo rm {} ";"`
 - 찾을 파일들을 명령어를 만들어 한번에 실행할 때 사용한다.
 - 실제로 삭제를 명령내릴때는 echo rm으로 명령을 한번 확인 한 후 삭제한다.
- awk {print}
 - `awk '{print $[N]}' {path}` : N번째 단어를 가져오는 방법

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1    localhost
255.255.255.255 broadcasthost
::1        localhost
/etc/hosts (END)
```

less /etc/hosts

```
▶ awk '{print $2}' /etc/hosts
Host
localhost
when
localhost
broadcasthost
localhost
```

awk '{print \$2}' /etc/hosts

- awk '{print \$[N] , \$[M]}' {path} : N번째 단어와 M번째 단어를 가져오는데 공백도 같이 가져옴

```
▶ awk '{print $2 $3}' /etc/hosts
HostDatabase
localhostis
whenthe
localhost
broadcasthost
localhost
```

awk '{print \$2 \$3}' /etc/hosts

```
▶ awk '{print $2, $3}' /etc/hosts
Host Database
localhost is
when the
localhost
broadcasthost
localhost
```

awk '{print \$2, \$3}' /etc/hosts

- diff
 - 2개의 파일을 한줄씩 비교
 - diff [file name 1] [file name 2]

텍스트 변환하기

- tr a-z A-Z
 - a-z를 A-Z로 변경
- tr -d '\t'
 - -d 옵션으로 문자 삭제

부모 프로세스와 자식 프로세스, 그리고 환경

부모프로세스와 자식프로세스

- 명령을 실행할때마다 자식프로세스 생성
 - 확인 방법
 1. cd /etc가 들어 있는 스크립트 생성
 2. ./[file name]으로 해당 스크립트 생성
 3. 현재 경로 확인
 4. 스크립트 실행
 5. 현재 경로 확인 → 현재 경로가 /etc가 아닌 것이 확인 가능
 - | (파이프라인)을 사용하면 파이프라인 하나당 자식 프로세스가 생성된다.
 - 자식 셸은

환경 변수

- 변수를 생성하면 해당 변수는 해당 인스턴스에서만 동작한다.
- printenv
- export를 이용한 환경변수 만들기
 1. 변수 생성
 2. export 변수명
- unset을 이용한 환경변수 삭제
 - unset [변수명]