

Chapter 3

UNIX Utilities for Power Users

1

YUAN LONG
CSC 3320 SYSTEM LEVEL PROGRAMMING
SPRING 2019

Updated based on original notes from Raj Sunderraman and Michael Weeks

What will be covered?

2

Section	Utilities
Filtering files	egrep, fgrep, grep
Programmable text processing	awk, sed
Sorting files	sort
Archiving files	tar
Searching for files	find
Switching users	su
Scheduling commands	crontab

Filtering files

3

- **grep, egrep, and fgrep**

- Filter out all lines that do not contain a **specified pattern** (i.e. output all lines containing a specified pattern)
- `grep -inw 'pattern' {fileName}*` **basic regular expression**
- `egrep -inw 'pattern' {fileName}*` **extended regular expression**
- `fgrep -inw 'pattern' {fileName}*` **fixed string**

-i	: ignore case
-n	: display line numbers
-w	: matches only the whole words

Check following options:

-E in grep
-x in egrep

grep Examples

4

- Assume we have a file called *grepfile*

```
$cat -n grepfile
```

```
1:Well you know it's your bedtime,  
2:So turn off the light,  
3:Say all your prayers and then,  
4:Oh you sleepy young heads dream of wonderful things,  
5:Beautiful mermaids will swim through the sea,  
6:And you will be swimming there too.
```

- Display the lines containing the pattern **/sw.*ng/**

```
$grep --color -n 'sw.*ng' grepfile
```

```
6:And you will be swimming there too.
```

- Display the lines containing the pattern **/a./**

```
$grep --color -n 'a.' grepfile
```

```
3:Say all your prayers and then,  
4:Oh you sleepy young heads dream of wonderful things,  
5:Beautiful mermaids will swim through the sea,
```

--color:
highlight the
matched string

grep Examples

5

grep pattern	Lines that match
<code>.nd</code>	3:Say all your prayers and then, 4:Oh you sleepy young heads dream of wonderful things, 6: And you will be swimming there too.
<code>^nd</code>	6: And you will be swimming there too.
<code>sw.*ng</code>	6:And you will be swimming there too.
<code>[A-D]</code>	5: B eautiful mermaids will swim through the sea, 6: A nd you will be swimming there too.
<code>\.</code>	6:And you will be swimming there too.
<code>a.\$</code>	5:Beautiful mermaids will swim through the sea a ,
<code>[a-m]nd</code>	3:Say all your prayers and then,
<code>[^a-m]nd</code>	4:Oh you sleepy young heads dream of wonderful things, 6: And you will be swimming there too.

egrep Examples

6

- Assume we still use the *grepfile* file

```
$cat -n grepfile
```

```
1:Well you know it's your bedtime,  
2:So turn off the light,  
3:Say all your prayers and then,  
4:Oh you sleepy young heads dream of wonderful things,  
5:Beautiful mermaids will swim through the sea,  
6:And you will be swimming there too.
```

- Display the lines containing the pattern **/sw.*ng/**

```
$egrep --color -n 'sw.*ng' grepfile
```

```
6:And you will be swimming there too.
```

- Display the lines containing the pattern **/s.+w/**

```
$egrep --color -n 's.+w' grepfile
```

```
4:Oh you sleepy young heads dream of wonderful things,  
5:Beautiful mermaids will swim through the sea,
```

egrep Examples

7

egrep or grep -E **pattern**

s.*w

4:Oh you **sleepy young heads dream of** wonderful things,
5:Beautiful mermaids**s will swim** through the sea,
6:And you will be **swimming** there too.

s.+w

4:Oh you **sleepy young heads dream of** wonderful things,
5:Beautiful mermaids**s will swim** through the sea,

Off|will

2:So turn **off** the light,
5:Beautiful mermaids **will** swim through the sea,
6:And you **will** be swimming there too.

im*ing

6:And you will be sw**imming** there too.

im?ing

<No Matches>

Programmable Text Processing

8

- ***sed, awk***

Powerful tools for editing

Examples:

- Remove selected lines
- Print a range of lines
- Modify specific words in the files
- <http://docstore.mik.ua/orelly/unix/sedawk/>

How do **sed** and **awk** work?

9

- Performs an **action** on all lines that match a particular **condition**
- The **action** could be from command line or a file

```
$sed '1,100 s/A/a/' test.txt
```

Action in a command line

```
$sed -f sedfile test.txt
```

Action in a file

- **Note:** in default both *sed* and *awk* do not modify the input file; without condition, the action will be performed on all lines.

Stream Editor (*sed*)

10

- Condition: select the matched lines
 - Line number, line range, regular expression(BRE in default)
 - ✦ E.g. 1,4 10,\$ /Expr/
- Action : things you can do with *sed*
 - Delete lines **d**
 - Print lines **P or p**
 - Substitution **s/old/new/g**
 - Insert lines **i**
 - Change lines **c**

```
sed '1,100 s/A/a/' test.txt
Condition: 1,100 Action: s/A/a
```

Delete and Print Lines

11

- \$ sed '/a/d' file > file.new
 - Deletes all lines containing 'a'
- \$ sed -n '/a/ p' file >new
 - Print lines containing 'a'
- \$ sed -n '200,300 p' file >new
 - Print line 200 to line 300

Try without -n, what will be printed?

-n: suppress the automatic printing of pattern space

Substituting Text

12

- `$ sed 's/^/ /' file > file.new`
 - indents each line in the file by 2 spaces
- `$ sed 's/^ *//' file > file.new`
 - removes all leading spaces from each line of the file
- `$ sed '200,300 s/A/a/g' f1 f2 f3 >new`
 - combine file f1, f2 and f3 together
 - replace 'A' with 'a' from line 200 to 300 in the new combined file
 - store the output of sed command to file new
- `$ cat f1 f2 f3 | sed '200,300 s/A/a/g' > new`

Inserting Text

13

- Add two lines at the beginning of file

```
$ cat -n dummy
```

```
1:one
2:two
3:three
4:four
5:five
6:six
```

```
$ cat sed1
```

```
1i\
Copyright 2016 by Yuan\
All rights reserved
```

```
$ sed -f sed1 dummy
```

```
Copyright 2016 by Yuan
All rights reserved
one
two
three
four
five
six
```

Replacing Text

14

- Replace lines 1-3 by “Lines 1-3 are censored”

```
$ cat dummy
```

```
1:one
```

```
2:two
```

```
3:three
```

```
4:four
```

```
5:five
```

```
6:six
```

```
$ cat sed2
```

```
1,3c\
```

```
Lines 1-3 are censored
```

```
$ sed -f sed2 dummy
```

```
Lines 1-3 are censored
```

```
four
```

```
five
```

```
six
```

Deleting Text

15

- Delete only those lines that contain 'o'

```
$ cat dummy
```

```
1:one
```

```
2:two
```

```
3:three
```

```
4:four
```

```
5:five
```

```
6:six
```

```
$ cat sed3
```

```
/. *o/d
```

```
$ sed -f sed3 dummy
```

```
three
```

```
five
```

```
six
```

```
$ sed '/.*o/d' dummy
```

```
three
```

```
five
```

```
six
```

awk Command

16

- ***awk*** [condition] [{action}]
- Condition
 - special tokens **BEGIN** or **END**
 - an expression involving logical operators, relational operators, and/or regular expressions
- Action: one of the following kinds of C-like statements
 - if-else; while; for; break; continue
 - assignment statement: var=expression
 - **print; printf;**
 - next (skip remaining patterns on current line)
 - exit (skips the rest of the current line)
 - list of statements

awk Command

17

- *awk* reads a line
 - breaks it into fields separated by tabs/spaces (in default)
 - or other separators specified by **-F option**
- Accessing individual fields
 - **\$1,...,\$n** refer to the values in fields 1 through n
 - **\$0** refers to entire line
- Example: Print the number of fields and first field in the `/etc/passwd` file.

```
$ awk -F: '{ print NF, $1 }' /etc/passwd
```

-F:	Use colon ':' as the field separator
------------	--------------------------------------

NF	Built-in variable, means number of fields
-----------	---

\$1	Refers to field 1
------------	-------------------

awk Command

18

- Special tokens in *awk*

BEGIN	Triggered before first line read
END	Triggered after last line read
FILENAME	Name of file being processed
NR	Current line #
NF	Number of fields

awk Example

19

- Now we are using a file “**passwd**” in the folder of **etc** under root directory

```
$cat /etc/passwd
```

```
nobody:*:-2:-2:Unprivileged User:/:usr/bin/false
```

```
root:*:0:0:System Administrator:/var/root:/bin/sh
```

```
...
```

```
lp:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
```

On each line,
there are 7 fields
separated by :

- Task 1: Only display the user name, user's home directory and login shell

1st field

6th field

7th field
- Task 2: Based on task 1, print out “Start of file” and “End of file” at beginning and end of the output separately.
- Task 3: Display the required information only for the **first two user**.
- Task 4: Based on task3, add the **line number** and **number of fields** at begging and end separately for each line.

NR

NF

awk Example : Task 1

20

- Task 1: Only display the **user name**, user's **home directory** and **login shell**

Assume the program for **awk** has been stored in **p1.awk**

```
$cat p1.awk
```

```
#Print out the first, sixth, and seventh fields in the remained lines  
{ print $1 " " $6 " " $7 }
```

- Execute *awk* command

```
$awk -F: -f p1.awk /etc/passwd
```

```
nobody / /usr/bin/false
```

```
root /var/root /bin/sh
```

```
...
```

```
lp /var/spool/cups /usr/bin/false
```

awk Example : Task 2

21

- Task 2: Based on task 1, print out “Start of file” and “End of file” at beginning and end of the output separately.

```
$cat p2.awk
#Before processing first line, print out "Start of file"
BEGIN { print "Start of file: " }
#Print out the first, sixth, and seventh fields in the remained lines
{ print $1 " " $6 " " $7 }
#After processing all lines, Print out "End of file" with Filename
END { print "End of file", FILENAME }
```

- Execute *awk* command

```
$awk -F: -f p2.awk /etc/passwd
Start of file:
nobody / /usr/bin/false
root /var/root /bin/sh
...
lp /var/spool/cups /usr/bin/false
End of file /etc/passwd
```

awk Example : Task 3

22

- Task 3: Display the required information only for the **first two users**.

```
$cat p3.awk
NR >= 1 && NR <= 2 { print $1 " " $6 " " $7 }
```

- Execute *awk* command

```
$awk -F: -f p3.awk /etc/passwd
nobody /usr/bin/false
root /var/root /bin/sh
```

awk Example : Task 4

23

- Task 4: Based on task3, add the ***line number*** and ***number of fields*** at beginning and end separately on each line.

```
$cat p4.awk
NR >= 1 && NR <= 2 { print NR $1 " " $6 " " $7 NF}
```

- Execute *awk* command

```
$awk -F: -f p4.awk /etc/passwd
1 nobody / /usr/bin/false 7
2 root /var/root /bin/sh 7
```

Sorting Files (sort)

24

- Sorts a file in ascending or descending order based on one or more fields.
- Individual fields are ordered lexicographically, which means that corresponding characters are compared based on their ASCII value.

Sorting Files (sort)

25

- `sort -tc -r {sortField -bfMn}* {fileName}*`
 - `-tc` separator is `c` instead of blank e.g. `-t:`
 - `-r` descending instead of ascending
 - `sortField` : `+POS1 [-POS2]` key positions start [up to end]
 - `-b` ignore leading blanks
 - `-f` ignore case
 - `-M` month sort (3 letter month abbreviation)
 - `-n` numeric sort

Sort Examples

26

\$ cat sort.dat

```
John Smith 1222 20 Apr 1956
Tony Jones 1012 20 Mar 1950
John Duncan 1111 20 Jan 1966
Larry Jones 1223 20 Dec 1946
```

0 1 2 3 4 5

\$ sort +0 -2 sort.dat

```
John Duncan 1111 20 Jan 1966
John Smith 1222 20 Apr 1956
Larry Jones 1223 20 Dec 1946
Tony Jones 1012 20 Mar 1950
```

\$ sort +4 -5 -M sort.dat

```
John Duncan 1111 20 Jan 1966
Tony Jones 1012 20 Mar 1950
John Smith 1222 20 Apr 1956
Larry Jones 1223 20 Dec 1946
```

\$ sort +4 -5 sort.dat

```
John Smith 1222 20 Apr 1956
Larry Jones 1223 20 Dec 1946
John Duncan 1111 20 Jan 1966
Tony Jones 1012 20 Mar 1950
```

Note: the position of field for sort starts from 0

Archiving (tar)

27

- Create a “tape archive” format file from the file list
 - `tar -cvf tarFileName fileList`
- Extract files from a “tape archive” format file to current directory
 - `tar -xvf tarFileName`
- Show the content of a “tape archive” format file
 - `tar -tvf tarFileName`

-f enables you to give a tar file name
Default name is /dev/rmto
-v verbose

Create a tar file

28

```
$ tar -cvf ch6.tar ch6
ch6/
ch6/menu.csh
ch6/junk/
ch6/junk/junk.csh
ch6/junk.csh
ch6/menu2.csh
ch6/multi.csh
ch6/expr1.csh
ch6/expr3.csh
ch6/expr4.csh
ch6/if.csh
ch6/menu3.csh
```

- Check the existence of ch6.tar

```
$ ls -l ch6.tar
-rw-rw-r-- 1 raj raj 20480 Jun 26 20:08 ch6.tar
```

Show Contents in a Tar File

29

```
$ tar -tvf ch6.tar
```

```
drwxr-xr-x raj/raj      0 2007-06-03 09:57 ch6/
-rwxr-xr-x raj/raj     403 2007-06-02 14:50 ch6/menu.csh
drwxr-xr-x raj/raj      0 2007-06-03 09:57 ch6/junk/
-rwxr-xr-x raj/raj    1475 2007-06-03 09:57 ch6/junk/junk.csh
-rwxr-xr-x raj/raj    1475 2007-06-03 09:56 ch6/junk.csh
-rw-r--r-- raj/raj     744 2007-06-02 15:59 ch6/menu2.csh
-rwxr-xr-x raj/raj     445 2007-06-02 15:26 ch6/multi.csh
-rwxr-xr-x raj/raj     279 2007-06-02 15:18 ch6/expr1.csh
-rwxr-xr-x raj/raj      98 2007-06-02 15:20 ch6/expr3.csh
-rwxr-xr-x raj/raj     262 2007-06-02 15:21 ch6/expr4.csh
-rwxr-xr-x raj/raj     204 2007-06-02 15:22 ch6/if.csh
-rw-r--r-- raj/raj     744 2007-06-02 16:01 ch6/menu3.csh
-rw-rw-r-- raj/raj      29 2007-06-21 11:06 date.txt
```

Extract a Tar File

30

- Assume we remove the original **ch6** folder first

```
$ rm -fr ch6
```

- Then extract the **Tar** file to the current folder

```
$ tar -xvf ch6.tar
```

```
ch6/  
ch6/menu.csh  
ch6/junk/  
ch6/junk/junk.csh  
ch6/junk.csh  
ch6/menu2.csh  
ch6/multi.csh  
ch6/expr1.csh  
ch6/expr3.csh  
ch6/expr4.csh  
ch6/if.csh  
ch6/menu3.csh  
date.txt
```

Searching files (find)

31

- ***find*** <startingDirectory> <matching criteria and actions>
 - Searching the files matching given expression starting from **pathName**
- Expression
 - -name pattern
 - ✦ true if the file name matches pattern
 - -perm oct
 - ✦ true if the octal description of file's permission equals oct
 - -type ch
 - ✦ true if the type of the file is ch (b=block, c=char ..)
 - -user userId
 - ✦ true if the owner of the file is userId
 - -group groupId
 - ✦ true if the group of the file is groupId

Find Examples

32

- `$ find / -name '*.java'`
 - searches for all Java file in the entire file system
- `$ find . -name 'sed*'`
 - Searches for all files with names starting “sed”

Substituting User

33

- **% su userName**
 - If userName is not specified, root is assumed
 - Need access privileges for this
 - Requires password
- **% sudo command**
 - User can execute command as super user
 - Requires password

Review

34

- Pattern matching (grep)
- Pattern matching and processing (awk,sed)
- Sort
- Archiving(tar)
- Searching files(find)