Exercises-Filters2

In this exercise we will practice with the Unix filters **cut**, and **tr**. We will also practice using **paste**, even though, strictly speaking, it is not a filter. In addition, we will expand our use of **grep** to add the options -**c, -v, -e, -i, -n**, and **-l** and the use of simple regular expressions. Last we will do a handful of real-life problems to put these tools together to do real work.

There is a lot of work in this exercise set. Do enough of the problems to master a filter, then move on to the next one. Do not ignore the final part where all the filters are used together. This is the most important part.

For the duration of these exercises, you will be using test files from the directory **samples/Data** beneath the public data area on hills. You will also need to create temporary files from time to time, so you can either get personal copies of the files you need and work in your own area, or you can work in the **Data** directory and place your temporary files in your home directory. This is what the answer key does. Instead of data files, some of the problems use the output of various Linux commands as input to work on.

Part One - cut/paste

Description

The filter **cut** slices its input vertically either by character position or using a delimiter and a field number. **cut** cannot rearrange fields. If you want to generate some output that takes fields #3 and #5 of one file and displays them as field #5 followed by field #3 you must cut the fields out individually and then use **paste** to paste them together. If you are using fields, the default delimiter for both **cut** and **paste** is a tab.

Review your class notes and text for the syntax of the **cut** and **paste** commands, or look at their manual pages.

Unless otherwise directed, you should just display the results of your commands on the screen.

Exercises

1. Look at the output of the **date** command. Using cut and column positions, create a file that contains the month and another that contains the day of the month. Then output the date as day month with a space between the day and the month.

Look at the file **samples/Data/Hired_Data** beneath the class public directory. Its format is **Name:Dept:Job** Then use a command to output

2. the **Names** only
3. the **Names** and **Jobs**, separated by a **:**
4. the **Jobs** then **Names**, separated by a **:** (this takes several commands)

5. Look at the file **Emp_Data**. Output the **Names** field as **First Last** rather than **Last, First**
6. Look at the files **st2** and **st3**. Note that they are different lengths. Output a list that has the first field of **st3** followed by the third field of **st2**, keeping the same delimiter. How was the difference in length resolved?

Part Two - tr

Description

The **tr** command translates, deletes, complements and squeezes characters. We will cover all of these except complement. (Complement is very useful, especially in conjunction with the other options, however. Try the following command on a text file:

cat file | tr -cs '[[:alnum:]]' '[\012*]')

tr needs one or two character strings to specify the characters that it is to translate. A shorthand to enumerating these characters is to use a character set. Thus, the character string **'abcde'** could be

specified as **'[a-e]'**. Just like wildcards, you can also use a character class in a character set. Character sets (and classes) are part of modern regular expressions, although there are a few minor differences we will discuss later. Here are the classes:

class	*meaning*	*class*	*meaning*
upper	any uppercase character	**alpha**	any alphabetic character
lower	any lowercase character	**digit**	any number
alnum	any alphabetic or digit	**punct**	any punctuation character
space	any whitespace character	**ascii**	any ASCII character
print	any printable character	**xdigit**	any hexadecimal digit

Examples:

tr -d '[[:alpha:][:punct:]]' - The set contains all characters that are members of either the class **alpha** or **punct**. (Delete all control and punctuation characters)

tr '[[:upper:]]' '[[:lower:]]' - lowercase all uppercase characters.

One more shortcut is helpful: to extend the size of the second set to match the first by duplicating the last element, simply put the last string in a character set and follow it by *****

tr 'aeiou' '[-*]' - replace all vowels by dashes

Exercises

1. Using **echo**, send **tr** the string "**UPPER lower**", telling it to delete all blanks. (Those are blanks between **UPPER** and **lower**)
2. Redo the last command, telling **tr** to squeeze out repeated blanks.
3. Tell it to squeeze out repeated blanks and change those remaining to pound symbols (**#**)
4. Last, squeeze out leading blanks, then translate uppercase characters to lowercase characters. (this is making use of the fact that tr can translate a sequence of characters to another sequence using a 1-for-1 substitution)
5. Examine the file **sorttest**. Output the file on the screen after changing the **#** delimiter to a comma.

Look at an **ls -l** listing of a directory. You want to cut out the size and name field, but **cut** relies on a single delimiter between fields. Let's design this solution:

6. take the **ls -l** output and squeeze successive blanks to a single one.
7. now add **cut** so that only size and name is output.

Part Three - grep

This part covers the **grep** command, adding options **-e -n** and **-l** The later ones use simple regular expressions. The files can be found in the directory **samples/Data** beneath the public data area.

Using **grep** only, display the lines in the file **u2** that

1. start with **cow**
2. start with the word **It**
3. contain exactly (consist of) **cow**
4. contain either **cow** or **animal**
5. contain both **cow** and **animal**, anywhere on the line.
6. output the lines in **u2** that start with **cow** with the line number in the file
7. output the number of lines in **u2** that contain **cow**
8. output the names of the files in the **Data** directory that contain **cow**

9. output the number of lines in each file in **Data** that contain **cow**
10. output the number of lines in **u2** that don't contain **cow**
11. output the lines of the **passwd** file whose shell field (the last field) is **/usr/bin/bash**
12. output the lines of **passwd** whose shell field is neither **/usr/bin/bash** nor **/usr/bin/ksh**
13. output the lines of **passwd** whose login field (the first field) is three characters long.
14. output the lines of **st.bad** that have an empty field

Part Four

This last section has problems that put all the filters together to solve real-world problems. Some of them are challenging and are good sources of questions for the Google Group.

Note that we are only using simple regular expressions in this exercise set, so you must do a bit more work on some of the problems.

1. Output the login (field #1) of all entries in **passwd** whose default group (field 4) is **200**.
2. Output a list of the different shells used in the file **passwd** (the last field)
3. how many different default groups (field #4) are there in **passwd**?
4. How many members (members are in field #4, separated by commas) are there in the line in the **group** file whose group id (field #3) is **3021**?
5. Create a file (in your home directory) named **E14dept** with only the names (field 2) of each person in the file **Emp_Manager1** whose dept field (field 3) is **E14**. The list should be sorted by the person's id number (field 1)
6. Student accounts on hills have the default group of **506**. Output the number of lines of **/etc/passwd** whose default group (field 4) is *not* **506** (i.e., who are not students). Compare this to the number of students in the file. Any predictions?

In the directory /p**ub/cs/gboyd/cs160a/filters2** there is a set of dummy files and directories. Use it for the following problems:

7. List the names of all objects that are readable by group.
8. List the names of the files (only) that are both readable by group and by other.
9. List the names of the directories (only) that are not executable by other.

Answers

Part One

1.
```
date | cut -c5-7 > ~/mon
date | cut -c9-10 > ~/day
paste -d' ' ~/day ~/mon
rm ~/day ~/mon
```
2.
```
cut -d: -f1 Hired_Data
```
3.
```
cut -d: -f1,3 Hired_Data
```
4.
```
cut -d: -f1 Hired_Data > ~/names
cut -d: -f3 Hired_Data >~/jobs
paste -d: ~/jobs ~/names
rm ~/jobs ~/names
```
5.
```
cut -d, -f1 Emp_Data > ~/last
cut -d: -f1 Emp_Data | cut -d, -f2 > ~/first
paste -d' ' ~/first ~/last
rm ~/first ~/last
```
6.
```
cut -d# -f1 st3 > ~/1
```

```
cut -d# -f3 st2 > ~/3
paste -d# ~/1 ~/3
rm ~/[13]
```

Part Two

1. `echo "UPPER     lower" | tr -d ' '`
2. `echo "UPPER     lower" | tr -s ' '`
3. `echo "UPPER     lower" | tr -s ' ' | tr ' ' '#'`
4. `echo "UPPER     lower" | tr -s ' ' | tr '[[:upper:]]' '[[:lower:]]'`
5. `tr '#' ',' < sorttest`
6. `ls -l | tr -s ' '`
7. `ls -l | tr -s ' ' | cut -d' ' -f5,9-`
   (the trailing - gets filenames with blanks but has a problem with symlinks)

Part Three

1. `grep "^cow" u2`
2. `grep "^It" u2`
3. `grep "^cow$" u2`
4. `grep -e "cow" -e "animal" u2`
5. `grep "cow" u2 | grep "animal"`
6. `grep -n "^cow" u2`
7. `grep -c "cow" u2`
8. `grep -l "cow" *`
9. `grep -c "cow" *`
10. `grep -cv "cow" u2`
11. `grep ':/usr/bin/bash$' passwd`
12. `grep -v -e ':/usr/bin/bash$' -e ':/usr/bin/ksh' passwd`
13. `grep '^...:' passwd`
14. There are three possibilities for an empty field: the first field (`^#`), the last field(`#$`), or a middle field(`##`):
    `grep -e '^#' -e '#$' -e '##' st.bad`

Part Four

1. To do this with simple REs, you first must remove all non-pertinent information. All we need to answer the question is the default group and the login. Then look for the correct group, and extract the logins:
   `cut -d: -f1,4 passwd | grep ':200$' | cut -d: -f1`
2. `cut -d: -f7 passwd | sort -u`
3. `cut -d: -f4 passwd | sort -u | wc -l`
4. `cut -d: -f3,4 group | grep '^3021:' | cut -d: -f2 | tr ',' '\n' | wc -l`
5. `sort -k1,1n Emp_Manager1 | cut -d: -f2,3 | grep ':E14$' | cut -d: -f1 > ~/E14dept`
6. `cut -d: -f4 /etc/passwd | grep -c '^506$'` compare this to `wc -l /etc/passwd`. At the time of this writing (end of summer), there were about 8900 students and 200 non-students

The tricky part of these last ones is preparing the data. You only need the permissions fields and the names field, but to cut them out you must fix the delimiters by sqeezing out extra spaces like this:

```
ls -l | tr -s ' ' | cut -d' ' -f1,9-
```

Note that we added 9 *through the end* in case a filename contained spaces. All of the commands below have this sequence as part:

7. `ls -l | tr -s ' ' | cut -d' ' -f1,9- | grep '^....r' | cut -d' ' -f2-`
8. `ls -l | tr -s ' ' | cut -d' ' -f1,9- | grep '^-...r..r' | cut -d' ' -f2-`
9. `ls -l | grep '^d' | tr -s ' ' | cut -d' ' -f1,9- | grep -v '^.........x' | cut -d' ' -f2-`