Exercises - Basic Regular Expressions

The data files used in these exercises are in the directory **`/pub/cs/gboyd/cs160a/samples/Data`** on *hills*. Make sure you examine the data file, run your command, and examine your output carefully to determine if your command works correctly.

This exercise set has answers at the back. Use them to check your work.

All parts of this exercise set require basic regular expressions (BREs), and do not require 'turning on' the extended regular expression operators or using the **`-E`** option.

Begin by reviewing the Basic Regular Expressions below:

General Rules

BREs are understood by every Unix command that understands regular expressions, particularly **`grep`**, **`sed`**, **`more`** and **`vi`**.

- **Always quote your regular expressions**. For our class, use single-quotes.
- **Regular expressions can match any part of the line.** If you want to control this, use anchors
- **Dont confuse regular expressions with shell wildcards**. Regular expressions are used by one of the commands above to match text. Shell wildcards are used by the shell to match filenames. If you quote your regular expressions, the shell will not confuse them with a wildcard.

Consider the file **`t`** below:

```
$ cat t
abc
bc
abc1d
abcd12
```

Operator	Matches	Examples using the file **`t`** above
`.` (period)	any single character	**`grep '...'`** matches all but the second line of **`t`**
`*`	0 or more *of the preceding character* (The character to the left of the **`*`**) If **`*`** is the first character in the RE, it matches a literal **`*`**	**`*` is a repetition operator.** It repeats the character before it **`grep 'c*d'`** matches any line with a d (0 or more **`c`**'s followed by a **`d`**) **grep 'c.*d'** matches the last two lines. (**`c`** followed by 0 or more of any character followed by a **`d`**)
`[1d]`	one character that is **`1`** or **`d`**	**`grep '[1d]'`** matches the last two lines
`[[:class:]]`	one character that is a member of *class*. Commonly-used *class*es are **`alpha`**, **`digit`**, **`space`**, **`upper`**, **`lower`**, **`alnum`**, **`punct`**	**`grep '[[:digit:]]'`** matches the last two lines **`grep '[[:digit:]][[:digit:]]'`** matches the last line **`grep '[[:digit:]][[:alpha:]]'`** matches the third line
`[^abc]`	one character that is any except **`a`** or **`b`** or **`c`**	**`grep '[^d]'`** matches every line (since each line has a character that is not **`d`**) **`grep '[^d]$'`** matches all except the third line **`grep '[^[:alpha:]]'`** matches the last two lines. (Lines that have a non-alphabetic character.)
`^` **`$`**	anchors. ^ matches the beginning-of-line. **`$`** matches the end-of-line.	**`grep '^a'`** matches all but the second line **`grep 'c$'`** matches the first two lines **`grep '[[:digit:]]$'`** matches the last line.

Part One

Using the file **`input1`**, write commands to output only the lines with the following characteristics:

1. that contains the word **`hello`** anywhere on the line
2. that start with the word **`hello`**
3. that start with any number (any digit)
4. that ends with the word **`hello`**
5. that ends with any alphabetic letter (upper- or lower- case) or a question mark
6. that ends with a period (be careful here).
7. that contains only the word **`hello`** (it's the only thing on the line)
8. that contain only numbers
9. that contain only numbers, dashes and space characters.
10. containing more than 9 characters (at least 10 characters. A character can be anything)
11. that start with any whitespace character
12. that contain a string. This is anything within double quotes. Allow empty strings like **`""`**
13. repeat the last command, but do not allow empty strings.
14. a phone number. This is three digits followed by a dash followed by four digits. Notice that this outputs phone numbers with area codes as well.
15. This time your phone number should *not* have an area code - only the three digit, dash, four digit local phone number. (You can assume that your phone number is preceded by a whitespace character.)
16. Last, allow your phone number to be seven consecutive digits as well as the three digit dash four digit type.

Part Two

In this part we use a delimited file named **`Depts`**. It is in the **`samples`** directory discussed above. Look at the file **`Depts`**. Its format is **`DeptID:DeptName:EmpID:EmpName`** The **`EmpID`** is an integer.

Write commands to output only the lines with the following characteristics:

1. the **`DeptID`** begins with an **`E`**
2. the **`DeptID`** has exactly two digits
3. The **`DeptName`** starts with **`M`**
4. The **`DeptName`** is more than one [alphabetic] word. The words can be separated by multiple spaces.
5. The **`EmpID`** is three digits

Part Three

In this part we will practice with matching lines from other delimited files. The first file, named **`sorttest`**, uses the **`'#'`** character as the delimiter and it has five fields. Start by examining the **`sorttest`** file in the **`samples`** directory. Notice that each field has a different format. This, coupled with which field we are interested in, enables us to make *simplifying assumptions* when working problems. (We will assume the **`sorttest`** file is much larger, and this is just a representative sample, so we must be conservative about our assumptions.)

Example:

Output the lines whose last field is **`Administrator`** (exactly).

Solution:

Since we are interested in the last field, we know that the last field is preceded by **`#`** and followed by the end of the line. We can use these facts to write a simple RE:

```
grep '#Administrator$' sorttest
```

1. Output lines whose third field is **`D14`**
2. Output lines whose first field is a three digit number.
3. Output lines whose next-to-last field has at least one uppercase letter in it

Next we will use a standard system file, the **`/etc/passwd`** file, to do a few more interesting problems. Take a look at this file using **`tail /etc/passwd`**. You will see lines that look like this:

```
gboyd:x:3496:208:Unix/Linux Guy:/users/gboyd:/bin/bash
```

where the fields are **`username`**, **`pass`**, **`userid`**, **`groupid`**, **`gecos`**, **`homedir`**, **`shell`**

We are going to combine our regular expressions with other tools to extract fields from records we specify.

4. Output the **`shell`** field of the user **`gboyd`**
5. Output the **`homedir`** field of the user **`cmetzler`**
6. Output the **`username`** field of the account with the **`userid 10025`**
7. Output all the **`username`**s whose **`groupid`** field is **`554`**
8. Output all the **`username`**s whose **`gecos`** field is empty
9. Output the **`username`** field of all users whose **`userid`** is five digits and whose shell is not **`/bin/bash`**

Answers

1. `grep 'hello' input1`
2. `grep '^hello' input1`
3. `grep '^[[:digit:]]' input1`
4. `grep 'hello$' input1`
5. `grep '[?[:alpha:]]$' input1`
6. `grep '[.]$' input1` (or, better, `grep '\.$' input1`) (Remember: `.` is an operator!)
7. `grep '^hello$' input1`
8. `grep '^[[:digit:]]*$' input1` (This will match empty lines. Can you fix it?)
9. `grep '^[[:digit:] -][[:digit:] -]*$' input1` (This matches lines with *only* 1 or more characters that are digits spaces or dashes. Use this example to fix the previous one.)
10. `grep '..........' input1` (If it contains more than 10 characters, it contains 10.)
11. `grep '^[[:space:]]' input1`
12. `grep '".*"' input1` or, better, `grep '"[^"]*"' input1`
13. `grep '"..*"' input1` or, better, `grep '"[^"][^"]*"' input1`
14. `grep '[[:digit:]][[:digit:]][[:digit:]]-[[:digit:]][[:digit:]][[:digit:]] [[:digit:]]' input1`
15. `grep '[[:space:]][[:digit:]][[:digit:]][[:digit:]]-[[:digit:]][[:digit:]] [[:digit:]][[:digit:]]' input1` (This is not perfect, as there can be more digits after the phone number.)
16. `grep -e '[[:space:]][[:digit:]][[:digit:]][[:digit:]]-[[:digit:]] [[:digit:]][[:digit:]][[:digit:]]' -e '[[:space:]][[:digit:]][[:digit:]] [[:digit:]][[:digit:]][[:digit:]][[:digit:]][[:digit:]]' input1` (This will be much easier with extended regular expressions.)

Part Two

In a colon(:)-delimited file, the regular expression `'[^:]*:'` can be used to skip the contents of a field. (It means *any number of non-colons, followed by a colon)*. Thus, `'^[^:]*:x'` is a regular expression that matches `x` at the start of the second field of a colon-delimited file.

1. `grep '^E' Depts`
2. `grep '^.[[:digit:]][[:digit:]]:' Depts` (`Deptid` starts with one alphabetic character.)
3. `grep '^[^:]*:M' Depts`
4. `grep '^[^:]*:[[:alpha:]][[:alpha:]]*  *[[:alpha:]]' Depts` (note: two spaces between the asterisks)
5. `grep ':[[:digit:]][[:digit:]][[:digit:]]:[^:]*$' Depts` (Matches a three-digit number in the next-to-last field.)

Part Three

1. Since the format of the third field is unique, all we need to do is specify the field delimiter on each side of our search string (to separate `D14` from `D144`, for example): `grep '#D14#' sorttest`
2. Since it is the first field, all we need do is specify the beginning-of-line on the left and the field delimiter on the right: `grep '^[[:digit:]][[:digit:]][[:digit:]]#' sorttest`
3. This is more difficult, as every field except the first can have an uppercase letter. The only solution here to restrict our match of an upper-case letter to the fourth field is to specify the entire line either starting on the left (the first through fourth fields) or on the right (the fourth and fifth fields). Of course,

the latter is shorter.

We are looking for an upper-case character `[[:upper:]]`. However, this can be in any position in the field, and to get to it we must *skip the other characters*. These characters can be anything *except the field delimiter*. An RE for a single character that is not `#` is `[^#]`, so we can specify [part of] the fourth field by `'[[:upper:]][^#]*#'` (The last `#` separates it from the fifth field.)

To distinguish the `#` in the RE above as the *fourth* `#` in the line, we must specify the last field. We don't care what is in it, so each character can be any character except `#`: `'[^#]*'` and it is followed by the end-of line. Thus our command is `grep '[[:upper:]][^#]*#[^#]*$' sorttest`

In each of the examples below, execute the command once* before *the cut command to see the result of the grep, then add the cut command when you are satisfied with the result.

4. This one is simple: specify the contents of the first field using the BOL anchor and delimiter. This isolates the correct line, then extract the field:

```
grep '^gboyd:' /etc/passwd | cut -d: -f7
```

5. Only the **`username`** and field number change:

```
grep '^cmetzler:' /etc/passwd | cut -d: -f6
```

6. This is a bit more difficult, as there are two internal fields that are integers. It looks like the **`userid`** field is preceded by a field that is always **`x`**. If this is reliable, we have a simple solution:

```
grep 'x:10025:' /etc/passwd | cut -d: -f1
```

However, if the use of the preceding field is *not* reliable, we must skip to the correct field

```
grep '^[^:]*:[^:]*:10025:' /etc/passwd | cut -d: -f1
```

7. Again, if you can make the simplifying assumption that the **`groupid`** field is numeric and the **`pass`** field cannot be, you have a simple solution:

```
grep '[[:digit:]]:554:' /etc/passwd | cut -d: -f1
```

If this is not a valid assumption, you must use the general solution

```
grep '^[^:]*:[^:]*:[^:]*:554:' /etc/passwd | cut -d: -f1
```

8. It looks like the *only* field that can be empty is the **`gecos`** field. If this is true, the solution is simple:

```
grep '::' /etc/passwd | cut -d: -f1
```

If this is not a valid assumption you have a bit of a mess again. Since the **`gecos`** field is field #5 of 7 it is easiest to specify the pattern from the far end of the record:

```
grep '::[^:]*:[^:]*$' /etc/passwd | cut -d: -f1
```

9. We will generalize the [simpler] solution where we searched for a specific **`userid`** before to get the records with 5-digit **`userid`**s:

```
grep 'x:[[:digit:]][[:digit:]][[:digit:]][[:digit:]][[:digit:]]:' /etc/passwd
```

This is very difficult, so we will introduce an *extended regular expression* here:

```
grep -E 'x:[[:digit:]]{5}:' /etc/passwd
```

(Note that these are all the student accounts, so the output is about 8000 lines.) Now the output of this command must be searched for lines whose **`shell`** is not **`/bin/bash`**. This is

```
grep -v ':/bin/bash$'
```

Putting it all together

```
grep -E 'x:[[:digit:]]{5}:' /etc/passwd | grep -v ':/bin/bash$' | cut -d: -f1
```

Interestingly, this semester, many of these accounts have the shell field **`/bin/drop`**. We probably want to exclude them:

```
grep -E 'x:[[:digit:]]{5}:' /etc/passwd | grep -v ':/bin/bash$' |
    grep -v ':/bin/drop$' | cut -d: -f1
```