NDG Linux Unhatched - NDG Linux Unhatched

N **content.netdevgroup.com**/contents/unhatched/400

Filtering Input,
Regular Expressions
And Basic Patterns

Basic Patterns

Regular expressions are patterns that only certain commands are able to interpret. Regular expressions can be expanded to match certain sequences of characters in text. The examples displayed on this page will make use of regular expressions to demonstrate their power when used with the `grep` command. In addition, these examples provide a very visual demonstration of how regular expressions work, the text that matches will be displayed in a red color.

Follow Along

Use the following `cd` command to change to the `Documents` directory.

sysadmin@localhost:~$ cd ~/Documents

The simplest of all regular expressions use only literal characters, like the example from the previous page:

sysadmin@localhost:~/Documents$ grep sysadmin passwd
sysadmin:x:1001:1001:System Administrator,,,,:/home/**sysadmin**:/bin/bash

Anchor Characters

Anchor characters are one of the ways regular expressions can be used to narrow down search results. For example, the pattern `root` appears many times in the `/etc/passwd` file:

sysadmin@localhost:~/Documents$ grep 'root' passwd
root:x:0:0:**root**:/**root**:/bin/bash
operator:x:1000:37::/**root**:

To prevent the shell from misinterpreting them as special shell characters, these patterns should be protected by strong quotes, which simply means placing them between single quotes.

The first anchor character ^ is used to ensure that a pattern appears at the *beginning* of the line. For example, to find all lines in /etc/passwd that *start* with root use the pattern ^root. Note that ^ must be the *first* character in the pattern to be effective.

sysadmin@localhost:~/Documents$ grep '^root' /etc/passwd
root:x:0:0:root:/root:/bin/bash

For the next example, first examine the alpha-first.txt file. The cat command can be used to print the contents of a file:

sysadmin@localhost:~/Documents$ cat alpha-first.txt
A is for Animal
B is for Bear
C is for Cat
D is for Dog
E is for Elephant
F is for Flower

The second anchor character $ can be used to ensure a pattern appears at the *end* of the line, thereby effectively reducing the search results. To find the lines that end with an r in the alpha-first.txt file, use the pattern r$:

**sysadmin@localhost:~/Documents$** grep 'r$' alpha-first.txt
B is for Bea**r**
F is for Flowe**r**

Again, the position of this character is important, the $ must be the *last* character in the pattern in order to be effective as an anchor.

Match a Single Character With .

The following examples will use the red.txt file:

sysadmin@localhost:~/Documents$ cat red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
read

One of the most useful expressions is the period `.` character. It will match any character except for the new line character. The pattern `r..f` would find any line that contained the letter `r` followed by exactly two characters (which can be any character except a newline) and then the letter `f`:

```
sysadmin@localhost:~/Documents$ grep 'r..f' red.txt
reef
roof
```

The same concept can be repeated using other combinations. The following will find four letter words that start with `r` and with `d`:

```
sysadmin@localhost:~/Documents$ grep 'r..d' red.txt
reed
read
```

This character can be used any number of times. To find all words that have at least four characters the following pattern can be used:

```
sysadmin@localhost:~/Documents$ grep '....' red.txt
reef
reeed
roof
reed
root
reel
read
```

The line does not have to be an exact match, it simply must *contain* the pattern, as seen here when `r..t` is searched for in the `/etc/passwd` file:

```
sysadmin@localhost:~/Documents$ grep 'r..t' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:1000:37::/root:
```

Match a Single Character With `[]`

The square brackets `[ ]` match a *single* character from the list or range of possible characters contained within the brackets.

For example, given the `profile.txt` file:

```
sysadmin@localhost:~/Documents$ cat profile.txt
Hello my name is Joe.
I am 37 years old.
3121991
My favorite food is avocados.
I have 2 dogs.
123456789101112
```

To find all the lines in the `profile.txt` which have a number in them, use the pattern `[0123456789]` or `[0-9]`:

```
sysadmin@localhost:~/Documents$ grep '[0-9]' profile.txt
I am 37 years old.
3121991
I have 2 dogs.
123456789101112
```

On the other hand, to find all the lines which contain any non-numeric characters, insert a ^ as the first character inside the brackets. This character *negates* the characters listed:

```
sysadmin@localhost:~/Documents$ grep '[^0-9]' profile.txt
Hello my name is Joe.
I am 37 years old.
My favorite food is avocados.
I have 2 dogs.
```

Note

Do not mistake [^0-9] to match *lines which do not contain numbers*. It actually matches *lines which contain non-numbers*. Look at the original file to see the difference. The third and sixth lines only contain numbers, *they do not contain non-numbers* so those lines do not match.

When other regular expression characters are placed inside of square brackets, they are treated as literal characters. For example, the . normally matches any one character, but placed inside the square brackets, then it will just match itself. In the next example, only lines which contain the . character are matched.

```
sysadmin@localhost:~/Documents$ grep '[.]' profile.txt
Hello my name is Joe.
I am 37 years old.
My favorite food is avocados.
I have 2 dogs.
```

Match a Repeated Character Or Patterns With *

The regular expression character * is used to match zero or more occurrences of a character or pattern preceding it. For example e* would match zero or more occurrences of the letter e:

```
sysadmin@localhost:~/Documents$ cat red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
read
sysadmin@localhost:~/Documents$ grep 're*d' red.txt
red
reeed
rd
reed
```

It is also possible to match zero or more occurrences of a list of characters by utilizing the square brackets. The pattern [oe]* used in the following example will match zero or more occurrences of the o character *or* the e character:

```
sysadmin@localhost:~/Documents$ grep 'r[oe]*d' red.txt
red
reeed
rd
rod
reed
```

When used with only one other character, * isn't very helpful. Any of the following patterns would match *every* string or line in the file: .* e* b* z*.

```
sysadmin@localhost:~/Documents$ grep 'z*' red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
read
```

```
sysadmin@localhost:~/Documents$ grep 'e*' red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
re
```

This is because * can match *zero* occurrences of a pattern. In order to make the * useful, it is necessary to create a pattern which includes more than just the one character preceding *. For example, the results above can be refined by adding another e to make the pattern ee* effectively matching every line which contains at least one e.

```
sysadmin@localhost:~/Documents$ grep 'ee*' red.txt
red
reef
reeed
reed
reel
re
```

Standard Input

If a file name is not given, the grep command will read from standard input, which normally comes from the keyboard with input provided by the user who runs the command. This provides an interactive experience with grep where the user types in the input and grep filters as it goes. Feel free to try it out, just press **Ctrl-D** when you're ready to return to the prompt.
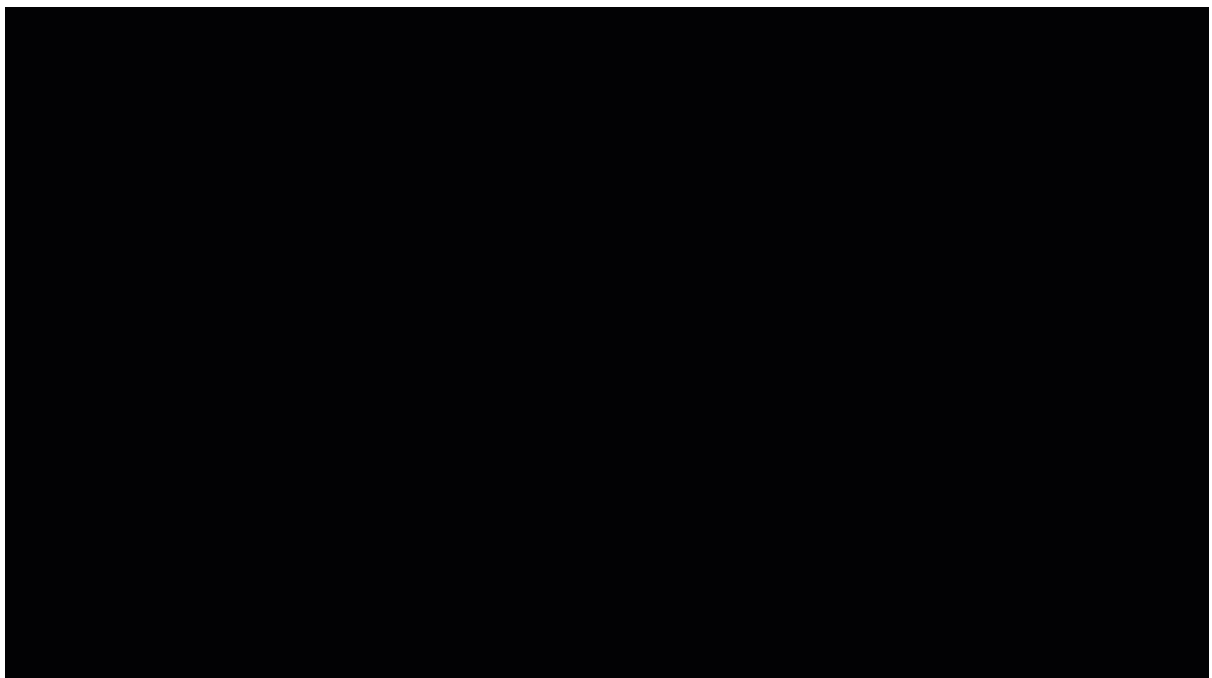
Follow Along

Use the following `cd` command to return to the home directory:

```
sysadmin@localhost:~/Documents$ cd ~
```


Linux is Open Source

Which makes it extremely versatile!