csce215 — UNIX/Linux Fundamentals
Fall 2021 — Lecture Notes: Where are my keys?

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

(6.1) Last time

Last time we learned about a tool called **make**, whose job is to help keep files up-to-date when their dependencies change.

Today, we will learn about some tools for **finding things**.

- **Regular expressions**, which can be used in many contexts to describe patterns to search for in text.
- Some more details about the `grep` command, whose job is to **look within files** for certain patterns.
- Several commands to **look for files** with certain properties, including `find` and `locate`.

(6.2) Finding files

`find`

Search for and act on files in or below a directory.

By default, `find` lists all files and directories.

```
$ find /usr/share/dict | head
/usr/share/dict
/usr/share/dict/cracklib-small
/usr/share/dict/words
/usr/share/dict/README.select-wordlist
/usr/share/dict/words.pre-dictionaries-common
/usr/share/dict/american-english
/usr/share/dict/british-english
```

We can add **tests** to filter out only certain files and directories, and **actions** to run commands on each thing that is found.

(6.3) Find: Tests

Use `-name` to look for files and directories whose names contain a certain string or wildcard pattern.

```
$ find /usr/share/dict -name *english*
/usr/share/dict/american-english
/usr/share/dict/british-english
```

Use `-type d` to look for only directories.

```
$ find /usr/share -type d | head -n 5
/usr/share
/usr/share/zoneinfo
/usr/share/zoneinfo/Etc
/usr/share/zoneinfo/Arctic
/usr/share/zoneinfo/Indian
```

Use `-type f` to look for only files.

```
$ find /usr/share/dict -type f
/usr/share/dict/cracklib-small
/usr/share/dict/README.select-wordlist
/usr/share/dict/american-english
/usr/share/dict/british-english
```

There are lots of other tests: modification types, file sizes, permissions, etc. See the `man` page.

(6.4) Find: Actions

Use `-print` to print the name of each match.

```
$ find /usr/share/dict -type f -print
/usr/share/dict/cracklib-small
/usr/share/dict/README.select-wordlist
/usr/share/dict/american-english
/usr/share/dict/british-english
```

Use -delete to delete each matching file.

```
$ find /usr/share/dict -type f -delete
find: cannot delete /usr/share/dict/cracklib-small: Permission denied
find: cannot delete /usr/share/dict/README.select-wordlist: Permission denied
find: cannot delete /usr/share/dict/american-english: Permission denied
find: cannot delete /usr/share/dict/british-english: Permission denied
```

Use -exec to execute a command for each match.

- {} for the name of the match
- \; to end the command

```
$ find /usr/share/dict -type f -exec head {} -n1 \;
007bond
Please use 'select-default-wordlist' superuser script
A
A
```

There are a few other actions, mostly for printing information or running commands. See the man page.

(6.5) Regular expressions

A **regular expression** ('regex') is

- a **sequence of characters** that
- describes a **set of strings**.

Main idea: A regular expression is a **pattern** that **matches** certain strings.

For example, the regular expression

```
[rR]\d-..
```

matches the string

```
R2-D2
```

(and many other strings), but does not match the string

```
C3-PO
```

(6.6) Who cares?

Regular expressions appear in a number of different places.

In tools you've seen already:

- In `vim`, the `/` command for search (and replace) accepts a regular expression.
- In `grep`, the pattern we search for is a regular expression.

Your favorite programming language supports regular expressions too!

- In Python, use the `re` module.
- In Perl, use the $=\sim$ operator.
- In Java, use the `java.util.regex` package.
- In C++, use the `regex` library.

(6.7) A quick clarification

Remember **wildcards** and **brace expansion**?

- `??.*`
- `*{Q,q}*`

These play a **somewhat similar role** to regular expressions, but are used in **different contexts**. They also **work differently** and are generally **less powerful**.

(6.8) Regular expressions in `grep`

We've seen `grep`, which looks for lines that match a given pattern, before.

`grep`	
Find lines that match a pattern.	
`-i` case insensitive	
`-v` find lines that *don't* match	
`-E` interpret special characters in patterns as regex operators	
`-r` search recursively in subdirectories	

When the pattern is more than just one specific string, we should use `-E` and use **single quotes** around the pattern.

Example: See words with a `q` followed by something other than a `u`:

```
$ grep -E 'q[^u]' /usr/share/dict/words
Chongqing
Chongqing's
Compaq's
Iqaluit
Iqaluit's
Iqbal
Iqbal's
Iraq's
Iraqi
Iraqi's
Iraqis
Qiqihar
Qiqihar's
Urumqi
Urumqi's
```

(6.9) Locating files

locate
List files on the system matching a pattern. --regex interpret the pattern as a regular expression

Example: Find -dev files for Python 3.4, 3.6, or 3.8:

```
$ locate --regex 'python3.[468]-dev'
/usr/share/doc/libpython3.8-dev
/usr/share/doc/python3.8-dev
/usr/share/lintian/overrides/libpython3.8-dev
/var/lib/dpkg/info/libpython3.8-dev:amd64.list
/var/lib/dpkg/info/libpython3.8-dev:amd64.md5sums
/var/lib/dpkg/info/python3.8-dev.list
/var/lib/dpkg/info/python3.8-dev.md5sums
```

(6.10) The simplest regular expressions

Most characters match themselves.

```
the
  ⇓
When the frost is on the punkin
```

A dot matches any character.

```
fo..er
  ⇓
and the fodder's in the shock
```

(6.11) Character classes

Use square brackets to give a **character class**.

```
y[aeiou][aeiou]
  ⇓
And you hear the kyouck and gobble
```

Use a dash inside the brackets to ask for any character in a given **range**.

```
[u-z]
  ⇓
of the struttin' turkey-cock
```

Use a caret inside the brackets to **negate** the class.

```
[^aeiuoAEIOU]
  ⇓
And the clackin' of the guineys
```

(6.12) Repetition

Use a question mark for **zero or one** of the previous.

```
t?he
  ⇓
and the cluckin' of the hens
```

Use a star for **zero or more** of the previous.

```
ro*
  ⇓
And the rooster's hallylooyer
```

Use a plus for **one or more** of the previous.

```
(t..)+
  ⇓
as he tiptoes on the fence
```

(6.13) Groups

Use parentheses to form groups.

```
([aeiuo][^aeiuo])+
  ⇓
O, it's then's the times a feller
```

(6.14) Either or

Use a pipe for either of two choices.

```
e+|his
  ⇓
is a-feelin' at his best
```

(6.15) Anchors

Use a caret to match the start of the line.

```
^[A-Za-z]+
  ⇓
With the risin sun to greet him
```

Use a dollar sign to match the end of the line.

```
[A-Za-z]+$
  ⇓
from a night of peaceful rest
```

(6.16) Not just for searching

sed
Stream editor for filtering and transforming text.
-e's/pattern/replacement/' Search and replace using a regular expression.

```
$ echo hello, world | sed -e's/h...o/goodbye/'
goodbye, world
```

(6.17) But wait, there's more!

These are the most important elements, common to most regex implementations. But there are loads of other features that can be used in regular expressions.

To learn more:

`https://regexone.com/`

To test and experiment with regular expressions:

`https://regex101.com/`