Ling 555 — Programming for Linguists

More unix basics & Regular Expressions

Robert Albert Felty

Speech Research Laboratory
Indiana University

Sep. 08, 2008

Network tools

- **ssh** secure remote login to another computer
- **sftp** secure file transfer to another computer (interactive)
- **scp** secure file transfer to another computer (non-interactive)
- **rsync** extremely powerful and smart file transfer (works both for local and remote computers — non-interactive)

Process management

ps	Display which processes are running (non-interactive)
top	Display which processes are running (interactive)
kill	Kill (abort) a process using the process ID
killall	Kill (abort) a process using the process name
nice	Set the cpu priority for a process
ionice	Set the disk usage priority for a process
nohup	Keep running after logging out
&	Run process in the background

Example

Run a long process in the background and don't hog system resources

```
nohup ionice -c2 -n7 nice -n 19 prog --progOpts &
```

Archiving and compressing

zip	Create a zip file
unzip	Extract contents from a zip file
gzip	Compress a file with GNU zip
gunzip	Decompress a file with GNU zip
bzip2	Compress a file with bzip compression (makes smaller files)
bunzip2	Decompress a file with bzip
tar	Create and extract tar archives Common uses:

create `tar -czvf file.tar.gz directory`

extract `tar -xzvf file.tar.gz`

Calculator

- **bc** Basic interactive calculator. Usually should invoke with the `-l` option
- **dc** Reverse polish style interactive calculator

Example

Add the first line of one file and the last of another

```
echo "`head -n1 numbers.txt` + `tail -n1
    numbers2.txt`" |bc -l
```

Example

Add the first 10 lines of a file (which contains one number per line)

```
echo "`head numbers.txt` + p" |dc
```

Environment variables

Definition

Most UNIX programs pay attention to environment variables, such as the language, timezone, and PATH. To see all currently set variables, type:

```
export
```

Example

To change a variable, do:

```
export PATH="/home/robfelty/bin:${PATH}"
```

Custom variables and aliases

Example

You can also create and use your own variables. If you frequently connect to the server speech.psych.indiana.edu, you can store that in a variable, e.g.

```
speech=speech.psych.indiana.edu
ssh $speech
```

Example

If you always want to have color listings, you can create an alias

```
alias ls='ls --color'
```

.rc files

Definition

Many UNIX programs, including the shell (we have been using the BASH shell), have files where one can store customizations between sessions.

Common .rc files

- .bashrc
- .vimrc
- .inputrc

Every time you open a new terminal, the .bashrc file is read.

Basic shell scripting

Definition

A shell script uses the exact same syntax as the command line shell you use (we have been using BASH). In this way, you can group commands together, to reduce work.

Basic shell scripting

Example

```
#!/bin/bash
# this script strips off any file extension from the
    argument, and runs the result through latex,
    bibtex, latex twice, dvips, ps2pdf, and then
    opens it with evince
SEED=`echo $1 | cut -f1 -d"."`
latex -interaction=batchmode $SEED && bibtex $SEED &&
     latex -interaction=batchmode $SEED && latex
    -interaction=batchmode $SEED && dvips -t letter
    -Ppdf $SEED.dvi -o $SEED.ps && ps2pdf $SEED.ps &&
     evince $SEED.pdf &
```

How might one improve this script?

```
1  #!/bin/bash
2  # this script syncs my school computer onto an external hard disk
       using rsync
3
4  # define a few constants
5  TARGET='/media/disk'
6  OPTIONS=' -avz --delete-after '
7  UMOUNT='FALSE'
8
9  echo "Executing incremental backup script"
10
11 # if /media/disk does not exist, create it, then mount the disk, and
       mark for unmounting
12 if [ ! -d /media/disk ]; then
13   echo "creating /media/disk and mounting"
14   UMOUNT='TRUE'
15   mkdir /media/disk
16   mount /dev/sdd1 /media/disk
17 fi
```

```
18 # first backup a few directories from the external disk to the local
       hard disk
19 ionice -c2 nice -n 19 rsync -avzu --exclude='.svn*'
       --exclude="*.swp"
       ${TARGET}/home/robfelty/{adam,RobsDocs,pics,R,matlab}
       /home/robfelty
20 ionice -c2 nice -n 19 rsync -avzu --exclude='.svn*'
       --exclude="*.swp" ${TARGET}/var/celex /var
21
22 #next backup everything from the local disk to the external
23 ionice -c2 nice -n 19 rsync $OPTIONS /selinux /bin /etc /home /lib
       /lib64 /misc /opt /root /sbin /usr /var ${TARGET}/ >
       ~/fedibbletyBackupLog.txt
24
25 if [[ $UMOUNT = 'TRUE' ]]; then
26   echo "unmounting and removing /media/disk"
27   umount /media/disk
28   rmdir /media/disk
29 fi
```

Line Endings

Definition

Mac, UNIX, and DOS (Windows) use different line ending characters, which can cause lots of problems

- `\r` Mac
- `\n` UNIX
- `\r\n` DOS

Converting between Mac, DOS, and UNIX

Most Linux distros ship with the programs *unix2dos* etc. Mac does not. Instead use the scripts provided in the `resources/utils` directory.

Common editors

nano (Open source version of pico).

Advantages:

- user-friendly. Lists commands at bottom of screen.
- small

Disadvantages:

- Not very powerful
- Not a default install on many UNIXes

vi Two-mode editor. This is my editor of choice.

advantages:

emacs Editor of choice for many programmers.

Swiss-army knife of editors.

Common editors

nano (Open source version of pico).

Advantages:

vi Two-mode editor. This is my editor of choice.

advantages:

- small (in size and memory usage)
- common (found on almost all UNIX systems by default)
- powerful (great regular expression support, and nice syntax highlighting)
- fast (your fingers never have to leave the home row. No mouse required)

Disadvantages:

- steep learning curve

emacs Editor of choice for many programmers.

Swiss-army knife of editors.

Common editors

nano (Open source version of pico).
Advantages:

vi Two-mode editor. This is my editor of choice.
advantages:

emacs Editor of choice for many programmers.
Swiss-army knife of editors. Advantages:

- Great syntax highlighting
- Single mode editor
- Includes all sorts of tools (news readers, e-mail readers, version control interfaces, friendfeed interface)

Disadvantages:

- Uses lots of memory
- Not a default install on many UNIXes

Globs (Wildcards)

Definition

Globs (wildcards) can be used by BASH, and by other programs (Microsoft Word & Excel) as shortcuts to match multiple expressions

*	Match zero or more characters.
?	Match any single character
`[...]`	Match any single character from the bracketed set. A range of characters can be specified with [-]
`[!...]`	Match any single character NOT in the bracketed set.
`{a,b,...}`	A list (set)

Globs (Wildcards)

N.B.

- An initial "." in a filename does not match a wildcard unless explicitly given in the pattern. In this sense filenames starting with "." are hidden. A "." elsewhere in the filename is not special.
- Pattern operators can be combined

Example

`chapter[1-5].*` could match *chapter1.tex*, *chapter4.tex*, *chapter5.tex.old*. It would not match *chapter10.tex* or *chapter1*

Using globs in BASH

Example

Delete all microsoft word documents in my home directory

```
rm -f ~/*.doc
```

Example

Convert all microsoft word documents in my home directory to plain text

```
for file in ~/*.doc; do antiword $file `basename
    $file .doc`.txt; done
```

Example

Create all files a-c with extensions txt,tmp,foo,bar

```
touch {a,b,c}.{txt,tmp,foo,bar}
```

Practice using globs in BASH

Download l55practiceFiles.tar.gz and untar it

1. Move all files ending in .txt to a new directory txt

   ```
   mkdir txt; mv *.txt txt
   ```

2. Copy files 10-19 to a new directory 10-19

   ```
   mkdir 10-19; cp 1[0-9] 10-19
   ```

3. list permissions for files ending in .txt which do not contain numbers

   ```
   ls -l [a-zA-Z].txt
   ```

   OR

   ```
   ls -l [!0-9].txt
   ```

4. Separate files into different directories according to their extension

   ```
   mkdir {tmp,foo,bar,txt}
   for file in *.{tmp,txt,foo,bar}; do mv $file
   `echo $file| cut -f 2 -d '.'` /$file; done
   ```

Regular expressions

- Finding the information you need from the databases will require the use of regular expressions
- Regular expressions are a feature in many programming languages that allow one to search for a given string in a body of text, including the use of some special characters
- Problem: I want to find all CVC words in the English CELEX database
  Solution: `grep -E '\\\[CVC\]\\' celex.cd`
- Problem: I want to know how many words that start and end with the letter *k*
  Solution: `grep -iEc '\\k[a-z]*k\\' celex.cd`

character classes and anything

Special characters: `. ? + * [] {} () | ^ $ \`

`.` matches any character

`[]` matches any of the characters within the brackets e.g. `[a0]` matches both *a* and *0*

Several predefined shortcuts are also possible

`[a-z]` matches all lowercase letters

`[A-Z]` matches all uppercase letters

`[a-zA-Z]` matches all uppercase and lowercase letters

`[0-9]` matches all numbers

Quantifiers

Special characters: `. ? + * [] {} () | ^ $ \`

- `?` matches 1 or 0 of the preceding character, e.g. `colou?r` matches *color* and *colour*
- `+` matches 1 or more of the preceding character, e.g. `bug +off` matches *bug off*, *bug off*, but not *bugoff*
- `*` matches any number of the preceding character, e.g. `colou*r` matches *color*, *colour*, *colouur* and so on
- `{}` used to specify the number of times a character should be matched. Ranges are also possible.

Example

`a{2}`	matches only *aa*
`[a-z]{2}`	matches two lowercase letters, e.g. *ab*
`[a-z]{2,4}`	matches 2–4 lowercase letters, e.g. *al* or *foo*

Greediness

Special characters: `. ? + * [] {} () | ^ $ \`

Definition

By default, `*` and `+` are greedy, meaning that they match as much as possible. Often this is not the intended effect.

Example

I want to strip out html tags from a document. I use the following regular expression: `<.*>` This will match `<span class='foo'>`. But it will also match `<span class='foo'>some text I don't want to get rid of</span>`

Solution: use negative character classes: `<[^<>]*>`

In Perl and python, you can use `.*?` and `.+?`

Grouping

Special characters: `. ? + * [] {} () | ^ $ \`

- `()` used to group sequences. Useful especially for backreferences (more on that later), and
- `|` used as an or operator, e.g. `x|y` matches either *x* or *y*

Example

`(m|M)(in|ax)imum` matches *minimum, maximum, Minimum* and *Maximum*

Backreferences

Special characters:

`. ? + * [] {} () | ^ $ \`

Definition

`\1` is a backreference. You can use multiple backreferences of the form `\n` where n is the nth pair of parentheses in the expression.

Example

Say I want to find common typos involving duplicate words (such as *a a* or *the the*). I could write an expression like so `(a|the) \1`
which says "match either *a* or *the* followed by a space followed by whatever was matched in the parentheses"

The beginning, the end, and escaping

Special characters: `. ? + * [] {} () | ^ $ \`

- `^` matches the beginning of the string
  Within brackets, negates the pattern, e.g. `[^xy]` matches everything but *x* or *y*
- `$` matches the end of the string
- `\` is the escape character. When you want to use one of the special characters as a normal character, it must be preceded by `\`

Grep specific information

- grep will search a file on a line by line basis, and return any lines which contain the regular expression
- In the case of CELEX, we will take advantage of the fact that fields are separated by\

Grep options

Like many UNIX programs, grep has quite a few options available. For a complete list, type `man grep`

- -E extended regular expressions — allows us to use all the special characters
- -i ignore case
- -c simply print the number of matches
- -v invert match, i.e. return everything that does not match the expression

These can be used in conjunction with one another, e.g.

```
grep -icv 'dog' file
```

returns the number of lines that do not contain the word dog from the file 'file'.

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’

  ```
  \\st
  ```

- all words that end in ‘ing’

  ```
  ing\\
  ```

- word that begin with ‘st’ and ending with ‘ing’

  ```
  \\st[a-z]*ing\\
  ```

- all monosyllabic words

  ```
  \\\[[CV]+\]\\
  ```

- all disyllabic words

  ```
  \\\[[CV]+\]\[[CV]+\]\\
  ```

- Find all words with a frequency of greater than 23

Substitution

Definition

Not only can you use regular expressions to match strings, but you can also replace matched strings with other strings. The easiest way to do this is with the program *sed*. **By default, sed prints out the entire input, replacing any patterns with the specified replacements** The basic form is like so:

```
sed 's/match/replace/flags' < infile > outfile
```

Example

Input: *The blue man sat next to the green man.*

```
echo 'The blue man sat next to the green man.' |
sed 's/man/woman/g'
```

Output: *The blue woman sat next to the green woman.*

Backreferences in replacements

Definition

Backreferences can be used not only in patterns, but also in replacements. This allows one to use dynamic replacements.

Example

File replacement: I want to get rid of spaces in filenames, because they can cause problems with UNIX scripts. I can use sed.

```
mv "foo bar.txt" foo_bar.txt
for file in *; do mv "$file" `echo $file|sed -E 's/
    /_/g'`; done
```

More transformations

- `\l` Makes the following character lower case
- `\u` Makes the following character upper case
- `\L` Makes all following characters lower case
- `\U` Makes all following characters upper case

Example

```
echo "Minimum"|sed -r 's/(in|ax)imum/\u\1/'
OR
echo "Minimum"|perl -pe 's/(in|ax)imum/\u\1/'
```

Practice (1)

1. Display the second column of courseBackground.txt

   ```
   cut -f2 courseBackground.txt
   ```

2. Create a new file with only the first and third columns of courseBackground.txt

   ```
   cut -f1,3 courseBackground.txt >
   courseBackground13.txt
   ```

3. Combine the courseBackground.txt with the new file you just created

   ```
   paste courseBackground.txt courseBackground13.txt
   > combinedFile.txt
   ```

4. Sort the courseBackground.txt file by nickname, ignoring case (HINT: use `-t $'\t'`)

   ```
   sort -k 2,2f -t $'\t' courseBackground.txt
   ```

Practice (2)

1. Count the number of entries in the Devil's Dictionary

```
grep -Ec '^[A-Z]{2,},' devilsDictionary.txt
```

2. Print out the all the entries in the Devil's dictionary (not the definition)

```
grep -E '^[A-Z]{2,},' devilsDictionary.txt | cut
-f1 -d ','
```

3. Count the number of occurrences of the word *the* in the Devil's Dictionary

```
grep -Eic '( |[^a-z]|^)the([^a-z]|$)'
devilsDictionary.txt
```

4. Count the number of indefinite articles in the Devil's Dictionary

```
grep -Eic '( |[^a-z]|^)(a|an)( |[^a-z] |$)'
devilsDictionary.txt
```

Practice (2) details

1. Count the number of entries in the Devil's Dictionary

   ```
   grep -Ec '^[A-Z]{2,},' devilsDictionary.txt
   ```

   Each entry starts with a word in all caps, at the beginning of a line, following by a comma.

2. Print out the all the entries in the Devil's dictionary (not the definition)

   ```
   grep -E '^[A-Z]{2,},' devilsDictionary.txt | cut
   -f1 -d ','
   ```

   I use the same regular expression as in the preceding example, but instead of using the -c option to count the number of words, I print them to standout out, then I use the cut command to only print the entry (which ends in a comma).

Practice (2) details

3. Count the number of occurrences of the word *the* in the Devil's Dictionary

```
grep -Eic '( |[^a-z]|^)the([^a-z]|$)'
devilsDictionary.txt
```

Let's think of this regular expression in steps.

- We could start of with simply `'the'`. But this will include words like *theater*
- `'the[^a-z]`. This excludes words like theater, but will not match words at the end of the line
- `'the([^a-z]|$)`. Now we are matching the characters *the* followed either by a non-letter, or the end of the line. But this would still match words like *lathe*.
- `'( |[^a-z]|^)the([^a-z]|$)'` Now we are also making sure that 'the' is preceded by either a space, a non-alphabetic character, or the beginning of the line

Practice (2) details

4. Count the number of indefinite articles in the Devil's Dictionary `grep -Eic '( |[^a-z]|^)(a|an)( |[^a-z]|$)' devilsDictionary.txt`
   This is basically the same as finding *the*, except we need to search for either *a* or *an*.