CS214 - Advanced UNIX

Lecture 2
Basic commands and regular expressions

Ymir Vigfusson

Shell expansions

Let us first consider regular expressions that arise when using the shell (shell expansions).

- `*` matches everything, 0 or more characters.
  - `$ ls lec*` would print
    `lecture1.pdf lecture2.pdf lecture5.pdf`
  - `$ ls l*ure2*` would print `lecture2.pdf`

Shell expansions

Let us first consider regular expressions that arise when using the shell (shell expansions).

- `?` matches exactly a single character.
  - `$ ls lecture?.pdf` would print
    `lecture1.pdf lecture2.pdf lecture5.pdf`

Shell expansions

- `[...]` matches any letter inside the square brackets.
  - `$ ls [sl]ec*` would print
    ```
    section.txt lecture1.pdf
    lecture2.pdf lecture5.pdf
    ```
- Typing `[abcd...xyz]` is tedious, so ranges like `[a-z]` are allowed.
  - `$ ls lec*[1-4]*` would print
    ```
    lecture1.pdf lecture2.pdf
    ```

Shell expansions

- `[^...]` matches any letter **NOT** inside the square brackets.
  - `$ ls [^a-p]ec*` would print `section.txt`

Shell expansions

- `{...,...}` matches any phrase inside the comma-separated square brackets
  - `$ ls {sec,lec}*` would print

```
section.txt lecture1.pdf
lecture2.pdf lecture5.pdf
```

Shell expansions

- Use backslash `\` to match a literal `* ? [ ] {` or `}`.
- Backslashes also allow you to add spaces and special characters such as `\n`
  - `$ ls What\ is\ the\ Matrix\?*` gives `What is the Matrix?.avi`
  - Without the backslashes of the spaces, `ls` would think you were asking for multiple files.

Bash shell script

- Write the following into a file called '`script`'
  - ```
    #!/bin/bash
    echo Yo World
    ```
- Type `chmod +x script` to make it executable.
- .. and execute it by typing `./script` (this is because the current directory normally doesn't get searched)
    ```

Variables

- Variables are denoted by `$varname`, and set by `varname=...`
  - ```
    #!/bin/bash
    WORLD=Earth
    echo Yo $WORLD
    ```
- There is a ton of built-in variables (such as `$USER`) that you can explore by typing `export`.
    ```

Single Quotes

- Tons of backslashes look ugly. The shell won't look for special characters in strings enclosed by single quotes, `'`.
  - `$ ls 'What is the Matrix?'*` prints
    `What is the Matrix?.avi`
- In bash shell scripts, quotes can enclose multiple lines.
  - ```
    #!/bin/bash
    echo 'This is the first line
    and this is the second line'
    ```
    ```

Double Quotes

- Double quotes " allow special characters.
  - `echo '$USER \n lala'` prints
    `$USER \n lala`
  - `echo "$USER \n lala"` prints `ymir` and
    `lala`

Unix tools

- Basic file manipulation utilities.
- `cd <dir>` changes current directory to `dir`. Parent directory is '`..`'.
- `ls -al` lists files in current directory including hidden files in long format.
- `cp -ir <src> <dst>` copies `src` to `dst` recursively and prompts before overwriting. `mv` moves files instead of copying.

Unix tools

- Basic file manipulation utilities.
- `rm -i <file>` will ask before deleting the file.
- `rm -rf <dir>` will completely remove all files and folders in `dir` without asking. Beware of this command, there has been much regret surrounding it...

Unix tools

- Essential tools for text and data parsing in scripts.
- `cat`, `less`, `grep`, `sed`, `gawk`, `tr`, `sort`, `uniq`, `head`, `tail`, `wc`
- Some of these (namely `sed` and `gawk`) provide a whole language which we will briefly discuss.

cat, less, head and tail

- `cat <file>` simply outputs the contents of `file`.
- `head -n 20 <file>` returns the first 20 lines (default 10).
- `tail -n 20 <file>` does the same for the last 20 lines.
- `tail -f <file>` keeps outputting appended data.
- `less <file>` fits the output to the terminal and allows you to scroll.

tr (translate)

Translates characters from one set to the other.

- `tr aeiou AEIOU <file>` outputs the contents of the file with all vowels capitalized.
- `tr -d '!@#$%^&*' <file>` deletes the specified characters
- `tr [A-Z] [a-z] <file>` outputs a lowercase version of the file.

sort

Sorts the lines of a text file alphabetically.

- `sort -r u  <file>` sorts the file in reverse order and deletes duplicate lines.
- `sort -n -k 2 -t : <file>` sorts the file numerically by using the second column, separated by a colon.

wc (word count)

Shows the number of lines, words and bytes in a file

- `wc -l <file>` shows the number of lines in the file.

grep

- The purpose of `grep` is to print the lines that match a particular pattern.
- `grep password file` will print exactly the lines that contain 'password'.
- The pattern can be a more complicated regular expression.
- The command
  `grep -o '[[:alpha:]][[:alnum:]]*' <file>`
  returns all valid C keywords and variable names.

Regular expressions

Pattern matching with more sophisticated rules than before. Important differences to shell expansion, e.g. the use of `?` and `{ }`.

- `{3,6}` will match between 3 and 6 occurrences of the previous pattern.
- `*` matches 0 or more occurrences, `+` operator matches 1 or more, `?` matches 0 or 1.
- `.` matches a single character.

Regular expressions

- We can group blocks together using `( )`. Disjunction (*or*) is done using `|`.
- E.g. `(la|ba){2,4}` matches 2 to 4 repeats of either `la` or `ba`, e.g. `lalaba` and `babababa`.
- `^` matches the beginning of the line, `$` the end.
- E.g. `^Big number:[ \t]*[0-9]+$` matches
  `Big number:  4294967296`

grep (continued)

- `grep` additionally has convenient sets for things like alphanumeric characters.
- These are e.g. `[[:alpha:]]`, `[[:alnum:]]`, `[[:space:]]`, `[[:cntrl:]]` for letters, alphanumeric characters, white space (including tabs), and control characters.
- Again,
  `grep -o '[[:alpha:]][[:alnum:]]*' <file>`
  returns all valid C keywords and variable names.

grep (continued)

- grep has a number of useful options.
- `grep -i` ignores the case.
- `grep -A 20 -B 10` also prints the next 10 lines before, and 20 lines after each match.
- `grep -v -o -n` respectively inverts the match, shows only the matched substring (instead of the whole matching line), and displays the line number.

Simple script

```
#!/bin/bash
tr ' ' '\n' $1 | grep '^[[:upper:]]' | wc -l
```

This will count the number of words that start with a capital letter.

sed (stream editor)

- This is a whole language. Unfortunately we'll just cover the most frequently used commands.
- `sed 's/not guilty/guilty/g'` would be a nasty prank (replaces all instances of `not guilty`) with `guilty`.
- The `g` stands for global, otherwise this would only match once.
- `sed` allows even more advanced regular expressions.

sed (stream editor)

- `sed 's/\r//g'` removes all carriage returns (DOS/Windows format).
- `sed 's/^$//p'` doubles the number of empty lines (`p` prints the match).
- `sed 's/You have \([0-9]*\) messages/\1/'` just prints the number of messages.
- The `\1` is a reference to the pattern that matched the first pair of brackets.

gawk or awk

- Small programming language that is pattern matching driven.
- The basic structure of an awk program is:

```
pattern1 { commands }
pattern2 { commands }
...
```

- Each pattern is a regular expression. The commands get executed when something matches the pattern.

Simple gawk program

```
/[Vv]ertex/ { v++ }
/[Ee]dge/ { print "(" $2 "," $3 ")"; e++ }
END { print v,e > "/dev/stderr" }
```