CS6
Practical System Skills
Fall 2019 edition
Leonhard Spiegelberg
lspiegel@cs.brown.edu
Office hours by appointment only from now onwards

No lecture on 8th October

Midterm: 22nd October (in 3 weeks)
09.99 Recap

Last lecture:

- foreground vs. background processes
- creation of processes via fork / exec
- sending signals to processes via kill
- archiving and compression via tar, gzip, bzip2, ...
What would you tell (angry) tux:

Good or bad practice?

I never quit programs, I always kill them using `kill -9 (SIGKILL)!`
10 String processing

CS6 Practical System Skills
Fall 2019
Leonhard Spiegelberg lspiegel@cs.brown.edu
we can use bash parameter expansion to manipulate strings

${#variable}  get length of string variable

Example:

tux@cs6demo:~$ STRING="hello world"
tux@cs6demo:~$ echo ${#STRING}
11
10.01 A more complex example

⇒ \${variable:offset} and \${variable:offset:length} can be used to extract substrings

```bash
#!/bin/bash
STRING="sealion"
for i in `seq 0 $(( ${#STRING} - 1))`;
do
echo \${STRING:$i}
done
```

sealion@cs6demo:~$ ./substrings.sh
sealion
ealion
aliion
lion
ion
on
n
### 10.02 Substring Removal

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>${haystack#needle}</code></td>
<td>Delete shortest match of needle from front of haystack</td>
<td>shortest prefix</td>
</tr>
<tr>
<td><code>${haystack##needle}</code></td>
<td>Delete longest match from front of haystack</td>
<td>longest prefix</td>
</tr>
<tr>
<td><code>${haystack%needle}</code></td>
<td>Delete shortest match of needle from back of haystack</td>
<td>shortest suffix</td>
</tr>
<tr>
<td><code>${haystack%%%needle}</code></td>
<td>Delete longest match of needle from back of haystack</td>
<td>longest suffix</td>
</tr>
</tbody>
</table>

Single `%/#` for shortest match, double `%%/##` for longest match!

⇒ can use for needle wildcard expression!
10.02 substring removal - examples

get file extension (shortest matching prefix)

```bash
PATH="/home/tux/file.tar.gz"; echo ${PATH#*.}
```

get basename (longest matching prefix)

```bash
PATH="/home/tux/file.txt"; echo ${PATH##*/}
```

get parent (path) (shortest matching suffix)

```bash
PATH="/home/tux/file.txt"; echo ${PATH%/*}
```

remove file extension (longest matching suffix)

```bash
PATH="/home/tux/file.tar.gz"; echo ${PATH%%.*}
```

Note: the extension here is tar.gz! To get gz, use ##*.

green part gets removed!
10.03 substring substitution

⇒ use `${parameter/pattern/string}` to perform substitution of first occurrence of *longest match* of pattern in parameter to string

Example:

```bash
PATH="/home/tux/file.tar.gz"; echo ${PATH/tux/sealion} /home/sealion/file.tar.gz
```
10.03 substring substitution

⇒ with # or % matching occurs from front or back

sealion@cs6demo:$ VAR="abc/abc/abc";echo \${PATH/abc/xyz}

/xyz/abc/abc

sealion@cs6demo:$ VAR="abc/abc/abc";echo \${PATH/#abc/xyz}

/abc/abc/abc

sealion@cs6demo:$ VAR="abc/abc/abc";echo \${PATH/%abc/xyz}

/abc/abc/xyz

sealion@cs6demo:$ VAR="abc/abc/abc";echo \${PATH/#/\abc/xyz}

/xyz/abc/abc

First occurrence from left to right of abc is replaced with xzy.
No match found from start.
Match found from back and replaced.
Match found at start (\ to escape /).
10.03 substring substitution - example

⇒ rename all .htm files to .html files! (same for jpg to jpeg)

```bash
for file in `ls *.htm`; do mv $file ${file/%.htm/.html}; done
```

Note the % to replace the extension!
10.03 Case conversions

⇒ you can use the following commands to change the case for words

${string^^}$  ⇒ converts string to UPPER CASE
${string^}$   ⇒ converts first character to upper case
${string,,,}$ ⇒ converts string to lower case
${string,}$   ⇒ converts first character to lower case
10.03 Case conversion - examples

tux@cs6demo:~$ string='HeLlo WORLD!'
tux@cs6demo:~$ echo ${string^^}
    HELLO WORLD!
tux@cs6demo:~$ echo ${string^}
    HeLlo WORLD!
tux@cs6demo:~$ echo ${string,}
    heLlo WORLD!
tux@cs6demo:~$ echo ${string,,,}
    hello world!
From scripting off to some useful commands...
It's a pipe(d) world!
there are multiple commands to work with text files

think always of a text file as a collection of lines which are made up of words (separable by whitespace)

using  allows to combine commands/programs

piped programs also often called filters because they manipulate a character stream
10.04 word count

wc = word count

wc [OPTION]... [FILE]...

⇒ counts words (separated by whitespace) and returns number

Per default prints newline, word and byte count for each file

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l</td>
<td>--lines</td>
<td>print the newline counts</td>
</tr>
<tr>
<td>-m</td>
<td>--chars</td>
<td>print the character counts</td>
</tr>
<tr>
<td>-w</td>
<td>--words</td>
<td>print the word counts</td>
</tr>
</tbody>
</table>
when used with stdin, wc simply delivers a number!

```
tux@cs6demo:~$ wc text.txt
  3 14 76 text.txt
```

```
tux@cs6demo:~$ wc -l text.txt
  3 text.txt
```

```
tux@cs6demo:~$ wc -m text.txt
  76 text.txt
```

```
tux@cs6demo:~$ wc -w text.txt
  14 text.txt
```

```
tux@cs6demo:~$ cat text.txt | wc
   3 14 76
```

```
tux@cs6demo:~$ cat text.txt | wc -l
  3
```

```
tux@cs6demo:~$ cat text.txt | wc -m
  76
```

```
tux@cs6demo:~$ cat text.txt | wc -w
  14
```

format is `<number> <file>`

numbers formatted in columns

<table>
<thead>
<tr>
<th>text.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>tux loves seafood so much one of his all-time favourites is squid so yummy!</td>
</tr>
</tbody>
</table>
10.04 wc - example

⇒ widely used piping example:
How many files XZY are in a directory?

```
ls *.jpg | wc -l
```

```
ls *.jpg | wc -w
```

same result
There will be more tools!
10.05 uniq

uniq [OPTION]... [INPUT [OUTPUT]]

⇒ reports or omits repeated lines
⇒ scans through a file and looks for adjacent matching lines

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>--count</td>
<td>prefix lines by number of occurrences</td>
</tr>
<tr>
<td>-d</td>
<td>-repeated</td>
<td>only print duplicate lines</td>
</tr>
<tr>
<td>-D</td>
<td>--all-repeated</td>
<td>print all duplicate lines</td>
</tr>
<tr>
<td>-u</td>
<td>--unique</td>
<td>only print unique lines</td>
</tr>
</tbody>
</table>
### 10.05 uniq - examples

<table>
<thead>
<tr>
<th>sample.txt</th>
<th>uniq -c sample.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>2 apple</td>
</tr>
<tr>
<td>apple</td>
<td>1 peach</td>
</tr>
<tr>
<td>peach</td>
<td>1 apple</td>
</tr>
<tr>
<td>apple</td>
<td>1 banana</td>
</tr>
<tr>
<td>banana</td>
<td>1 mango</td>
</tr>
<tr>
<td>mango</td>
<td>2 cherry</td>
</tr>
<tr>
<td>cherry</td>
<td>1 apple</td>
</tr>
</tbody>
</table>

- **count duplicates across adjacent groups**

As always options can be combined!

<table>
<thead>
<tr>
<th>uniq -D sample.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
</tr>
<tr>
<td>apple</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>cherry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uniq -u sample.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>peach</td>
</tr>
<tr>
<td>banana</td>
</tr>
<tr>
<td>mango</td>
</tr>
<tr>
<td>apple</td>
</tr>
</tbody>
</table>

- **print groups with no duplicates**

- **print groups with duplicates**

- **print groups with duplicates as often as they occur**

- **count duplicates**

- **across adjacent groups**

- **As always options can be combined!**
10.05 sort

sort lines of text files

```
sort [OPTION]... [FILE]...
```

⇒ many options to tune sorting

⇒ sorts ascending per default, i.e. `a, b, c` instead of `c, b, a`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-r</code></td>
<td><code>--reverse</code></td>
<td>reverse result</td>
</tr>
<tr>
<td><code>-f</code></td>
<td><code>--ignore-case</code></td>
<td>ignore case while sorting</td>
</tr>
<tr>
<td><code>-n</code></td>
<td><code>--numeric-sort</code></td>
<td>to sort file numerically</td>
</tr>
</tbody>
</table>
### 10.05 sort - examples

#### lexical sort

<table>
<thead>
<tr>
<th>sample.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
</tr>
<tr>
<td>apple</td>
</tr>
<tr>
<td>peach</td>
</tr>
<tr>
<td>apple</td>
</tr>
<tr>
<td>banana</td>
</tr>
<tr>
<td>mango</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>apple</td>
</tr>
</tbody>
</table>

#### numeric sort

<table>
<thead>
<tr>
<th>numbers.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sort numbers.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1999</td>
</tr>
<tr>
<td>-3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sort -n numbers.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1999</td>
</tr>
<tr>
<td>-3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>97</td>
</tr>
<tr>
<td>200</td>
</tr>
</tbody>
</table>
10.05 word count revisited

`sort sample.txt | uniq -c`

⇒ `sort` lines, then counting for each adj. group yields word count!

<table>
<thead>
<tr>
<th>sample.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
</tr>
<tr>
<td>apple</td>
</tr>
<tr>
<td>peach</td>
</tr>
<tr>
<td>apple</td>
</tr>
<tr>
<td>banana</td>
</tr>
<tr>
<td>mango</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>cherry</td>
</tr>
<tr>
<td>apple</td>
</tr>
</tbody>
</table>

sort

```
apple
apple
apple
apple
apple
banana
cherry
cherry
cherry
mango
peach
```

uniq -c

```
4 apple
1 banana
2 cherry
1 mango
1 peach
```
fmt = format

⇒ can be used to format lines to specified width, i.e. justification

⇒ fmt -width to format text to width characters.
   At least one word per line.

⇒ Use fmt -l to split into words!
10.05 tr - translate

tr = translate

⇒ simple tool to replace characters

⇒ many more options under man tr

Useful example:

tr -d "[:blank:]" removes whitespace
10.05 word count revisited - again

⇒ what are the top 5 frequent words in Hamlet?

curl https://cs.brown.edu/courses/cs0060/assets/hamlet.txt \
| fmt -l hamlet.txt \
| tr -d "[:blank:]" \
| sort \
| uniq -c \
| sort -nr \
| head -n 5

Pipeline steps:
1. download text file
2. split text into words
3. remove whitespace surrounding words
4. sort words (creates groups for uniq)
5. count adjacent groups
6. sort reverse groups to get most frequent word
7. return top 5 words via head
10.06 Columnar files

⇒ many commands like `uniq -c` prints output in columns

⇒ CSV=comma separated values files or
   TSV=tab separated values
   offer "column" based storage of text data

⇒ data separated by a separator character (, or \t )

<table>
<thead>
<tr>
<th>csv file</th>
<th>tsv file</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnA,columnB,columnC</td>
<td>columnA    columnB    columnC</td>
</tr>
<tr>
<td>hello,12,4.567</td>
<td>hello  12  4.567</td>
</tr>
<tr>
<td>world,,8.9</td>
<td>world   8.9</td>
</tr>
</tbody>
</table>
10.07 CSV files

⇒ no standard, however, should follow "standardization" attempt under RFC-4180 [https://tools.ietf.org/html/rfc4180](https://tools.ietf.org/html/rfc4180)

⇒ separate fields using ,

⇒ rows separated using newline character

⇒ to escape comma or newline, quote field using "

⇒ escape " in quoted field using double quote
### 10.07 CSV files

#### Example:

```plaintext
<table>
<thead>
<tr>
<th>a-complicated-csv-file.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;this is a column containing &quot;&quot;quoted content&quot;&quot;&quot;, whitespace in a column is fine &quot;to escape NEWLINE this needs to be within &quot;&quot;, the same goes for ,!&quot;,42</td>
</tr>
</tbody>
</table>
```

Though this is not standardized, much data gets shared as CSV files...
### 10.07 working with columnar files

> **cut** allows you to remove or select parts from each line

**cut** OPTION... [FILE]...

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c</code></td>
<td><code>--characters=LIST</code> select only characters</td>
</tr>
<tr>
<td><code>-b</code></td>
<td><code>--bytes=LIST</code> select only these bytes</td>
</tr>
<tr>
<td><code>-d</code></td>
<td><code>--delimiter=DELIM</code> use DELIM instead of TAB for field delimiter</td>
</tr>
<tr>
<td><code>-f</code></td>
<td><code>--fields=LIST</code> select only these fields</td>
</tr>
<tr>
<td></td>
<td><code>--complement</code> select the complement</td>
</tr>
</tbody>
</table>

> **LIST** is a comma separated list of numbers and ranges, e.g. 2, 5–8

**bytes e.g. useful for binary files**
10.07 cut - examples

echo "Hello world" | cut -c 1,7-11
Hworld

echo "Hello world" | cut -f2 -d ' ' world

echo "Tux's secret is sealion123" | cut -d ' ' -f 1-3 --complement sealion123

!!! byte positions are numbered starting with 1 !!!

Note: for ASCII chars -b and -c yield the same result!
cut works over multiple lines!

- can use `cut` to extract columns

Examples:

- `cut -f<n> table.txt`  # extract n-th column
- `cut -f1 --complement table.txt`  # remove first column
- `cut -f1,3 table.txt`  # extract first and third column
### 10.07 Using `cut` & Co

**Example:** extract columns 1 and 3 from simple csv file and add a header

<table>
<thead>
<tr>
<th>example.csv</th>
<th>out.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone Pro, Apple, $999</td>
<td>Product, Price</td>
</tr>
<tr>
<td>Pixel 3, Google, $499</td>
<td>iPhone Pro, $999</td>
</tr>
<tr>
<td>Galaxy S10, Samsung, $644</td>
<td>Pixel 3, $499</td>
</tr>
<tr>
<td></td>
<td>Galaxy S10, $644</td>
</tr>
</tbody>
</table>

```bash
echo "Product,Price
`cut -f3,1 -d',' example.csv``" > out.csv
```

Note the newline here!
10.07 Some notes on `cut`:

⇒ `cut` ignores order of `LIST`, i.e. `cut -f1,3` is the same as `-f3,1`

⇒ Same goes for `-c`

⇒ Do not try to parse CSV files with `cut`, there are better tools.

⇒ Stick to manipulation of simple output, e.g. from other bash commands like `uniq -c`
10.07 Combining columns

⇒ paste allows to combine two files column-wise

paste - merge lines of files
paste [OPTION]... [FILE]...

⇒ -d parameter for delimiter

⇒ -s paste one file at a time, i.e. transposed result

⇒ can be used to reorder columns!
### 10.07 paste - basic examples

**Example: merging columns**

```
paste -d',' countries.txt capitals.txt
```

<table>
<thead>
<tr>
<th>countries.txt</th>
<th>USA</th>
<th>France</th>
<th>Italy</th>
<th>Brazil</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Washington</td>
<td></td>
<td>Rome</td>
<td>Brasilia</td>
</tr>
</tbody>
</table>

**Example: transposing columns**

```
paste -sd ',,' countries.txt capitals.txt
```

<table>
<thead>
<tr>
<th>countries.txt</th>
<th>Washington</th>
<th>Paris</th>
<th>Rome</th>
<th>Brasilia</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
many of the text processing commands expect a file(path) as parameter

writing to tmp files is cumbersome and does not allow for one-liners

< (cmd) allows to pass stdout of cmd like a filepath

> (cmd) allows to pass file param to stdin of cmd
echo "Product,Price
`cut -f3,1 -d',' example.csv`" > out.csv

cat <(echo "Product,Price") <(cut -f1,3 -d',' example.csv) > out.csv

treated like we would two file paths, i.e.
cat fileA.txt fileB.txt

There's always a one-liner around :)

10.08 Adding a header - revisited
### 10.08 paste - reordering columns

<table>
<thead>
<tr>
<th>table.txt</th>
<th>table31.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  10   20</td>
<td>20   A</td>
</tr>
<tr>
<td>B  11   21</td>
<td>21   B</td>
</tr>
<tr>
<td>C  12   22</td>
<td>22   C</td>
</tr>
</tbody>
</table>

```
paste <(cut -f3 table.txt) <(cut -f1 table.txt)> table31.txt
```
10.09 diff

diff [OPTION]... FILES

⇒ compares files line by line

⇒ –Y to put output in two columns for direct comparison

⇒ lines prefixed with < are from the first file,
   with > from the second file

⇒ exit status of 0 indicates that the files are the same

⇒ detailed add (a), change (c), delete (d) syntax to follow changes,
  e.g. 0a2 means after line 0 2 lines of … need to be added.
### 10.09 diff - example

Tux is a little penguin who loves to work with the shell under Ubuntu.

<table>
<thead>
<tr>
<th>storyA.txt</th>
<th>Tux is a proud penguin who loves to work with the shell on his Macbook.</th>
</tr>
</thead>
<tbody>
<tr>
<td>storyB.txt</td>
<td>Tux is a little penguin who loves to work with the shell under Ubuntu.</td>
</tr>
</tbody>
</table>

```bash
tux@cs6demo:~$ diff storyA.txt storyB.txt
1c1
< Tux is a little penguin
---
> Tux is a proud penguin
3c3
< under Ubuntu.
---
> on his Macbook.
```

```bash
tux@cs6demo:~$ diff -y storyA.txt storyB.txt
Tux is a little penguin who loves to work with the shell under Ubuntu.

<table>
<thead>
<tr>
<th>storyA.txt</th>
<th>storyB.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tux is a little penguin who loves to work with the shell under Ubuntu.</td>
<td>Tux is a proud penguin who loves to work with the shell on his Macbook.</td>
</tr>
</tbody>
</table>
```
10.09 diff - example w. process substitution

⇒ comparing two directories w.r.t to their structure

tux@cs6demo:~$ diff <(ls /usr) <(ls /usr/local)

1a2
> etc
5c6
< local
---

> man

add after line 1 from other file line2
xargs - build and execute command lines from standard input

xargs [options] [command [initial-arguments]]

⇒ allows you to execute a command multiple times by feeding words as arguments to it!

⇒ -a file to read from file, else stdin.

⇒ -n max-args use at most max-args per command line

⇒ many more options, as always man xargs
10.10 xargs - downloading urls

```
xargs -n 1 -a urls.txt curl -O
```

Alternatively:

```
cat urls.txt | xargs -n 1 curl -O
```

<table>
<thead>
<tr>
<th>urls.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cs.brown.edu/courses/cs0060/assets/slides/slides1.pdf">https://cs.brown.edu/courses/cs0060/assets/slides/slides1.pdf</a></td>
</tr>
<tr>
<td><a href="https://cs.brown.edu/courses/cs0060/assets/slides/slides2.pdf">https://cs.brown.edu/courses/cs0060/assets/slides/slides2.pdf</a></td>
</tr>
<tr>
<td><a href="https://cs.brown.edu/courses/cs0060/assets/slides/slides3.pdf">https://cs.brown.edu/courses/cs0060/assets/slides/slides3.pdf</a></td>
</tr>
<tr>
<td><a href="https://cs.brown.edu/courses/cs0060/assets/slides/slides4.pdf">https://cs.brown.edu/courses/cs0060/assets/slides/slides4.pdf</a></td>
</tr>
<tr>
<td><a href="https://cs.brown.edu/courses/cs0060/assets/slides/slides5.pdf">https://cs.brown.edu/courses/cs0060/assets/slides/slides5.pdf</a></td>
</tr>
</tbody>
</table>
10.11 More text commands

⇒ there are many more text processing commands available on *NIX, e.g.:

- join
- expand/unexpand
- look
- fold
- column
- iconv

⇒ large list under https://www.tldp.org/LDP/abs/html/textproc.html
Next lecture

Regular expressions

- grep
- sed
- awk

Homework 4 out today!

Lab today: Regex intro
End of lecture.
Next class: Thu, 4pm-5:20pm @ CIT 477