

# CS6

## Practical System Skills

**Fall 2019 edition**

Leonhard Spiegelberg  
[lspiegel@cs.brown.edu](mailto:lspiegel@cs.brown.edu)



# Logistics

---

⇒ 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, ...

## 09.99 Recap - quiz

---

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](mailto:lspiegel@cs.brown.edu)*

## 10.01 Basic string processing

---

⇒ 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

substrings.sh

```
#!/bin/bash
STRING="sealion"

for i in `seq 0 $(( ${#STRING} - 1 ))`;
do
    echo ${STRING:$i}
done
```



```
sealion@cs6demo:~$ ./substrings.sh
sealion
ealion
alion
lion
ion
on
n
```

## 10.02 substring removal

<code>\${haystack#needle}</code>	delete shortest match of needle from front of haystack	shortest prefix
<code>\${haystack##needle}</code>	delete longest match from front of haystack	longest prefix
<code>\${haystack%needle}</code>	delete shortest match of needle from back of haystack	shortest suffix
<code>\${haystack%%needle}</code>	delete longest match of needle from back of haystack	longest suffix

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*)

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



get basename (longest matching *prefix*)

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



get parent (path) (shortest matching *suffix*)

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



remove file extension (longest matching *suffix*)

```
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:

```
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
```

first occurrence from left to right of abc is replaced with xyz

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

```
/abc/abc/abc
```

no match found from start

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

```
/abc/abc/xyz
```

match found from back and replaced

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

```
/xyz/abc/abc
```


match found at start (\ to escape /)

## 10.03 substring substitution - example

---

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

```
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,}
```

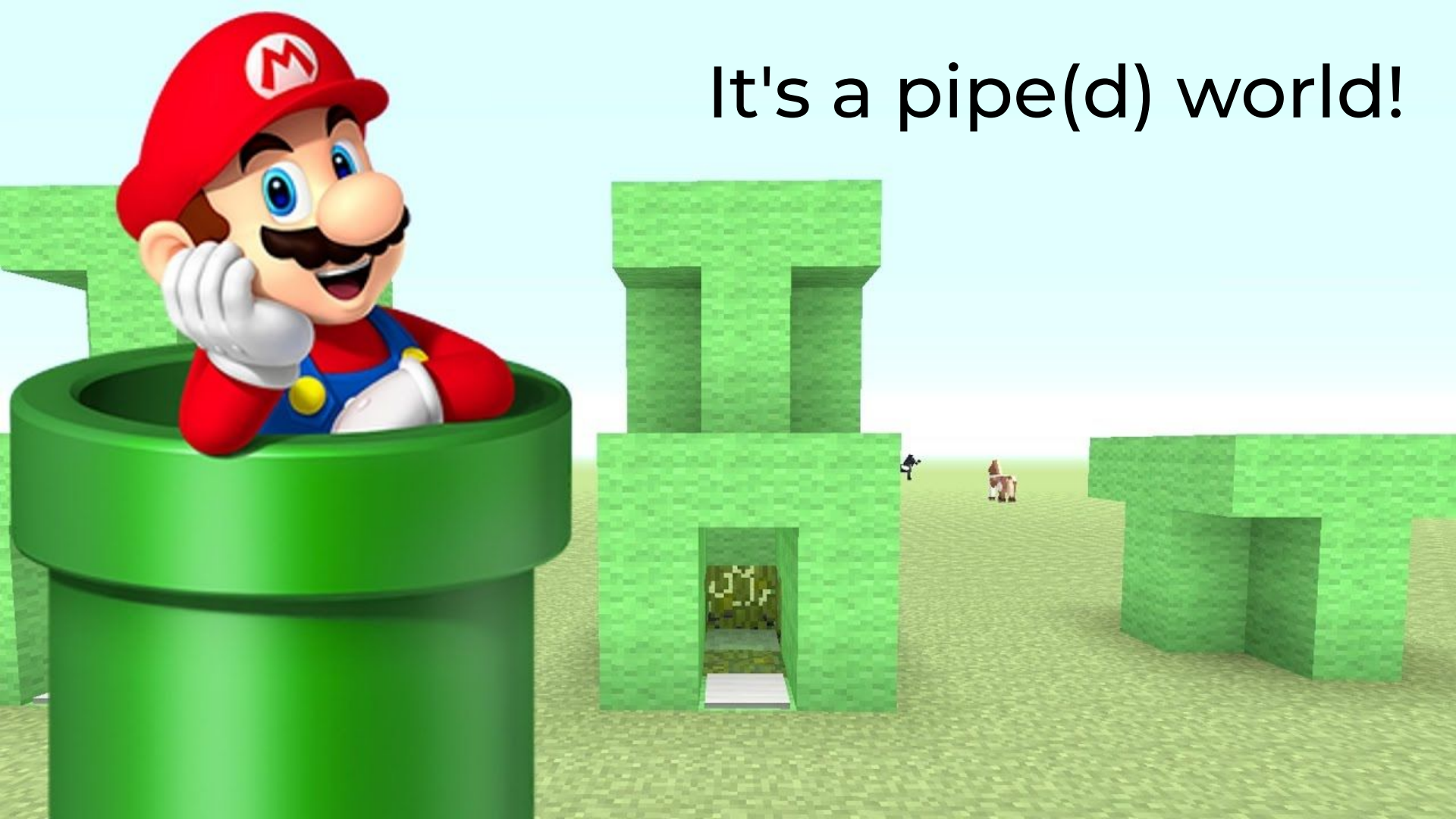
```
heLlo WORLD!
```

```
tux@cs6demo:~$ echo ${string,,}
```

```
hello world!
```

From scripting off to some useful  
commands...

It's a pipe(d) world!





## 10.04 Working with text files

---

- ⇒ 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

<code>-l</code>	<code>--lines</code>	print the newline counts
<code>-m</code>	<code>--chars</code>	print the character counts
<code>-w</code>	<code>--words</code>	print the word counts

## 10.04 wc - examples

⇒ 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

text.txt

tux loves seafood so much  
one of his all-time favourites is squid  
so yummy!

## 10.04 wc - example

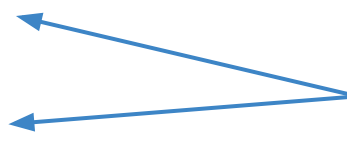
---

⇒ 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

---

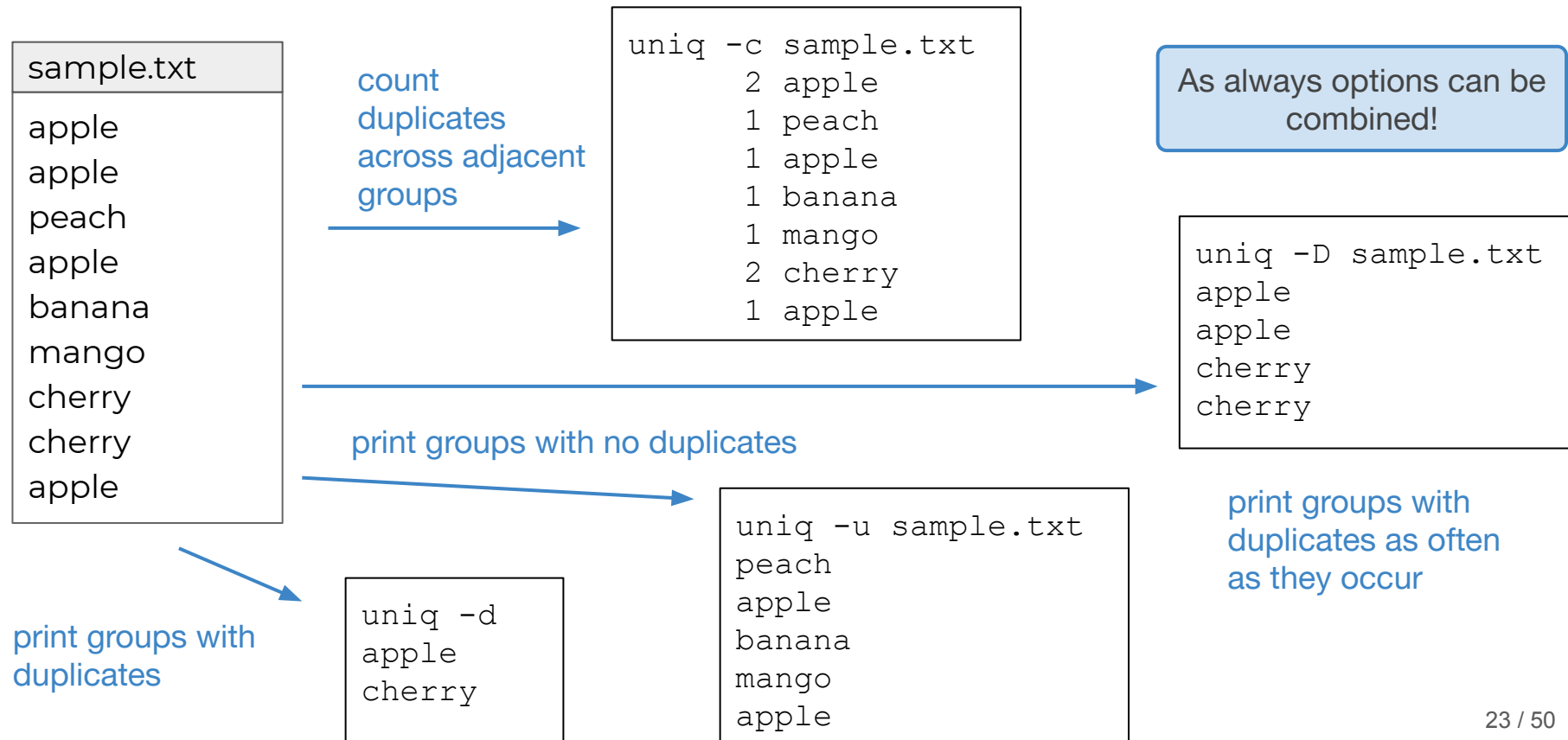
`uniq [OPTION]... [INPUT [OUTPUT]]`

⇒ reports or omits repeated lines

⇒ scans through a file and looks for adjacent matching lines

<code>-c</code>	<code>--count</code>	prefix lines by number of occurrences
<code>-d</code>	<code>-repeated</code>	only print duplicate lines
<code>-D</code>	<code>--all-repeated</code>	print all duplicate lines
<code>-u</code>	<code>--unique</code>	only print unique lines

## 10.05 uniq - examples



## 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

-r	--reverse	reverse result
-f	--ignore-case	ignore case while sorting
-n	--numeric-sort	to sort file numerically

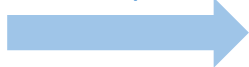


# 10.05 sort - examples

## lexical sort

sample.txt
apple
apple
peach
apple
banana
mango
cherry
cherry
apple

sort sample.txt



apple  
apple  
apple  
apple  
banana  
cherry  
cherry  
mango  
peach

## numeric sort

numbers.txt
34
2
65
200
97
-3
-1999

sort numbers.txt



-1999  
-3  
2  
200  
34  
65  
97

sort -n numbers.txt



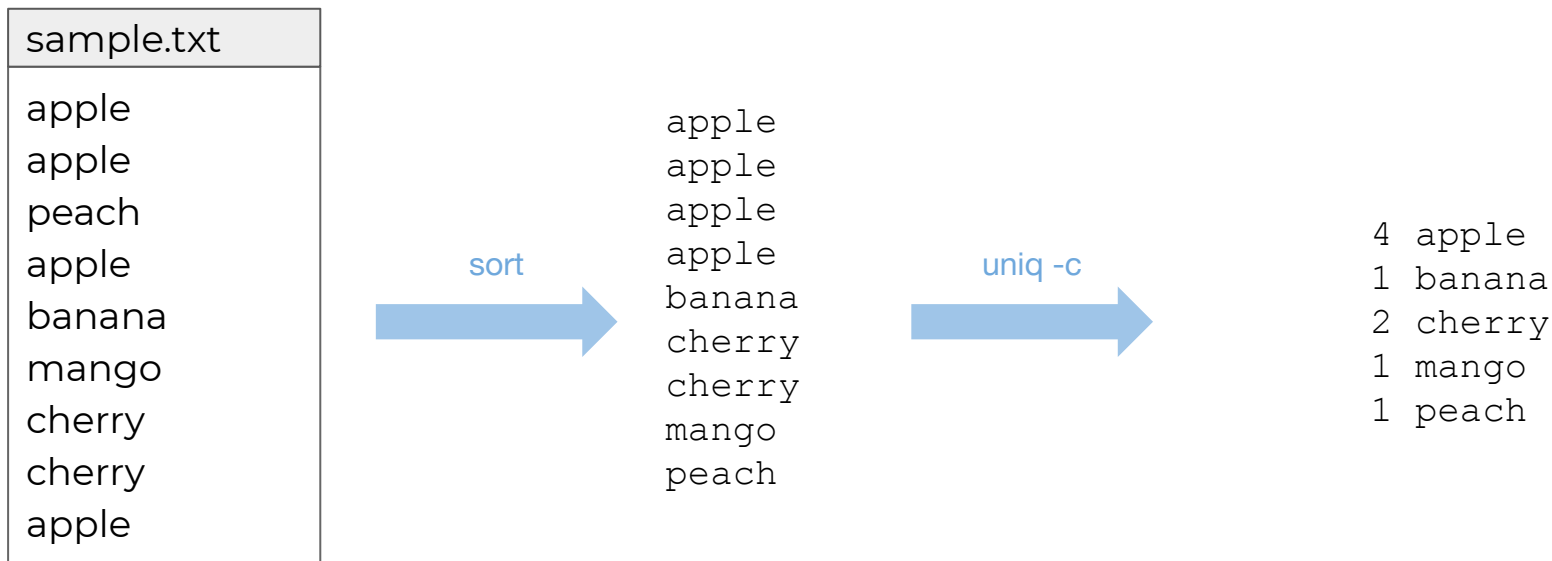
-1999  
-3  
2  
34  
65  
97  
200

## 10.05 word count revisited

---

```
sort sample.txt | uniq -c
```

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



## 10.05 splitting lines into words via `fmt`

---

`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 -1` 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



character class

## 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 -1 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 )

csv file

```
columnA,columnB,columnC  
hello,12,4.567  
world,,8.9
```

tsv file

```
columnA  columnB  columnC  
hello   12    4.567  
world    8.9
```

## 10.07 CSV files

---

- ⇒ no standard, however, should follow "standardization" attempt under RFC-4180 <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:

a-complicated-csv-file.csv

```
"this is a column containing ""quoted content""",whitespace in a column is fine  
"to escape  
NEWLINE  
this  
needs  
to be within "", the same goes for ,!,"42
```

Though this is not standardized, much data gets shared as CSV files...



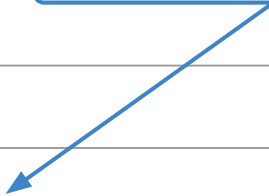
## 10.07 working with columnar files

---

⇒ `cut` allows you to remove or select parts from each line

`cut OPTION... [FILE] ...`

bytes e.g. useful for  
binary files



<code>-c</code>	<code>--characters=<u>LIST</u></code>	select only characters
<code>-b</code>	<code>--bytes=<u>LIST</u></code>	select only these bytes
<code>-d</code>	<code>--delimiter=<u>DELIM</u></code>	use DELIM instead of TAB for field delimiter
<code>-f</code>	<code>--fields=<u>LIST</u></code>	select only these fields
	<code>--complement</code>	select the complement

⇒ LIST is a comma separated list of numbers and ranges, e.g. 2, 5-8

## 10.07 cut - examples

---

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

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

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

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

Note: for ASCII chars -b and  
-c yield the same result!

## 10.07 cut - manipulating column!

---

⇒ `cut` works over multiple lines!

⇒ can use `cut` to extract columns

tab separated file




table.txt

A	10	20
B	11	21
C	12	22

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

example.csv

```
iPhone Pro,Apple,$999  
Pixel 3,Google,$499  
Galaxy S10,Samsung,$644
```

out.csv

```
Product,Price  
iPhone Pro,$999  
Pixel 3,$499  
Galaxy S10,$644
```

```
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  
USA,Washington  
France,Paris  
Italy,Rome  
Brazil,Brasilia
```

countries.txt

USA  
France  
Italy  
Brazil

### Example: transposing columns

```
paste -sd ' ' countries.txt capitals.txt  
USA,France,Italy,Brazil  
Washington,Paris,Rome,Brasilia
```

countries.txt

Washington  
Paris  
Rome  
Brasilia

## 10.08 Process substitution

---

⇒ 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`



## 10.08 Adding a header - revisited

---

```
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 paste - reordering columns

---

table.txt		
A	10	20
B	11	21
C	12	22

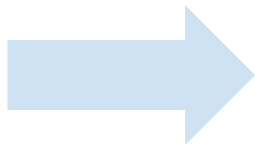


table31.txt	
20	A
21	B
22	C

```
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@cs6demo:~$ diff storyA.txt storyB.txt
```

```
1c1
< Tux is a little penguin
---
> Tux is a proud penguin
3c3
< under Ubuntu.
---
> on his Macbook.
```

a : add  
c : change  
d : delete

storyA.txt

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

storyB.txt

Tux is a proud penguin  
who loves to work with the shell  
on his Macbook.

```
tux@cs6demo:~$ diff -y storyA.txt storyB.txt
```

```
Tux is a little penguin      | Tux is a proud penguin
who loves to work with the  | who loves to work with the
under Ubuntu.                | on his Macbook.
```

## 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
```

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

bin		bin
	>	etc
games		games
include		include
lib		lib
local		man
sbin		sbin
share		share
src		src

add after line 1 from other file line2

## 10.10 xargs

---

`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
```

urls.txt

```
https://cs.brown.edu/courses/cs0060/assets/slides/slides1.pdf  
https://cs.brown.edu/courses/cs0060/assets/slides/slides2.pdf  
https://cs.brown.edu/courses/cs0060/assets/slides/slides3.pdf  
https://cs.brown.edu/courses/cs0060/assets/slides/slides4.pdf  
https://cs.brown.edu/courses/cs0060/assets/slides/slides5.pdf
```

## 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