Text Processing, Redirection and Pipes

Class assignment 2.1

- Create directory `classes` in your home dir
- Make sure that everyone can read `classes`, as well as your home directory
- Create subdirectory named `class02` in `classes/` and enter it
- Copy all files in `/share/classes/class02/` into the current directory

Class assignment 2.1

- Create directory `classes` in your home dir

```
mkdir classes
```

- Make sure that everyone can read `classes`, as well as your home directory

```
chmod a+rx . classes          (or just check permissions)
```

- Create subdirectory named `class02` in `classes/` and enter it

```
mkdir classes/class02  ; cd classes/class02
```

- Copy all files in `/share/classes/class02/` into the current directory

```
cp /share/classes/class02/* .
```

Text parsing in Linux

- Text is the standard interface in Linux
- This naturally seems inconvenient for us now that we are used to more intuitive interfaces involving graphics and touch.
- However, text interface is extremely powerful, because it allows all Linux commands to easily "interact" with one another in solving complex tasks.
- We will see today how this approach can be used to write simple "one-line programs" to solve the following tasks:
  - Print line *n* to line *m* of a given file
  - Take a phone directory file and create a count table for number of names that start in each letter of the alphabet
  - Find the 5^{th} most common word in a given text file

Lecture overview

- Text processing commands (**`head, tail, wc`**) ←
- Redirection
- Pipes
- Text delimited tables (`cut, paste, sort`)
- Simple character-based editing (`tr`)

Text processing commands

Text provides a simple mechanism for storing data and for passing it between applications and utilities.

Text is the main interface of UNIX-based systems.

Displaying file contents

- `cat file` – displays the contents of a (text) file
- `more file` – displays the contents of a (text) file up to what can be viewed in one screen – can be scrolled down
- `less file` – similar to `more`, but you can scroll up and down and search (as you do with `man` pages)

Viewing portions of a file – `head` and `tail`

- The **`head`** command shows only the first few lines of a file

```
head [options] [files]
```

- With no options, **`head`** shows first 10 lines

- Select command line options for **`head`**:
  - **`-nN`** – show first ***N*** lines
  - **`-n-N`** – show all but last ***N*** lines
  - **`-cN`** – show first ***N*** bytes
  - **`-c-N`** – show all but last ***N*** bytes

  (type **`head --help`** for all options)

Viewing portions of a file – `head` and `tail`

- The `tail` command shows only the **last** few lines of a file

```
tail [options] [files]
```

- With no options, `tail` shows **last** 10 lines

- Select command line options for `tail`:
  - `-nN` – show last ***N*** lines
  - `-n+N` – show all lines starting from line ***N***
  - `-cN` – show last ***N*** bytes
  - `-c+N` – show all bytes starting from byte ***N***
  - `-f` – used to view a log file as it is updated (not discussed this semester)

  (type `tail --help` for all options)

`head` and `tail` – examples

Given a file called '`phone_dir.txt`':

```
ADAMS, Andrew 7583
BARRETT, Bruce 6466
BAYES, Ryan 6585
BECK, Bill 6346
BENNETT, Peter 7456
GRAHAM, Linda 6141
HARMER, Peter 7484
MAKORTOFF, Peter 7328
MEASDAY, David 6494
NAKAMURA, Satoshi 6453
REEVE, Shirley 7391
ROSNER, David 6830
```

`head` and `tail` – examples

```
head -n3 phone_dir.txt
```

```
ADAMS, Andrew 7583
BARRETT, Bruce 6466
BAYES, Ryan 6585
```

```
tail -n3 phone_dir.txt
```

```
NAKAMURA, Satoshi 6453
REEVE, Shirley 7391
ROSNER, David 6830
```

```
head -c30 phone_dir.txt
```

```
ADAMS, Andrew 7583
BARRETT, Br
```

`head` and `tail` – examples

```
tail -n+12 phone_dir.txt
```

```
MEASDAY, David 6494
NAKAMURA, Satoshi 6453
REEVE, Shirley 7391
ROSNER, David 6830
```

```
tail -c50 phone_dir.txt
```

```
toshi 6453
REEVE, Shirley 7391
ROSNER, David 6830
```

```
tail -c11 phone_dir.txt
```

```
David 6830
```

Counting lines/words/bytes (chars) – `wc`

- The `wc` command counts the number of lines, words, and bytes in a file

```
wc [options] [files]
```

- Command line options for `wc`:
  - `-l` – print the number of lines
  - `-w` – print the number of words
  - `-c` – print the number of bytes

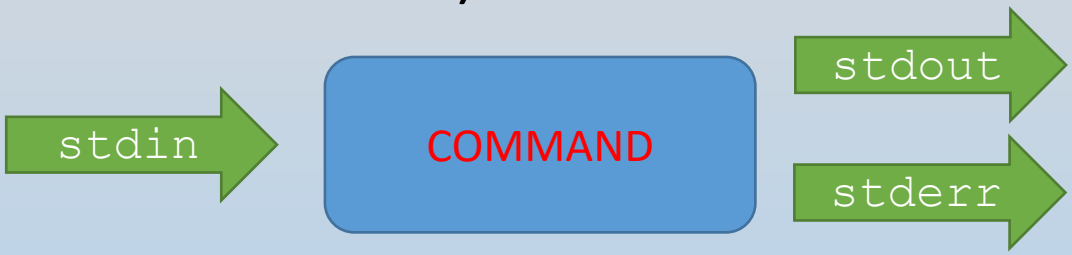
- With no options, all three are printed

Lecture overview

- Text processing commands (`head, tail, wc`)
- Redirection ←
- Pipes
- Text delimited tables (`cut, paste, sort`)
- Simple character-based editing (`tr`)

Input and output redirection

- Normally, when you run a command, it gets input from the keyboard (**stdin**) and sends output to the screen (**stdout** and **stderr**)



- Use '**>**' or '**1>**' to redirect **stdout** (standard output) to file
- Use '**2>**' to redirect **stderr** (standard error) to file
- Use '**&>**' to redirect **stdout** and **stderr** to file together
- Use '**<**' to redirect file into **stdin** (standard input) of command

Output redirection

- We use the '>' operator to redirect the output of a command
- For example, to create a file containing a listing of files in the current directory – redirect `stdout` via `>`:

```
ls > files
cat files
```

```
file1.txt
file2
file2.tmp
file2.txt
file3.txt
```

Output redirection – `stdout` and `stderr`

- Example of standard output and standard error

```
ls *.c *.txt
```

```
ls: cannot access *.c: No such file or directory   ← stderr
file1.txt   file2.txt   file3.txt                  ← stdout
```

- Redirection of standard output (standard error directed to screen)

```
ls *.c *.txt > files          ← redirect stdout
```

```
ls: cannot access *.c: No such file or directory   ← stderr
```

```
cat files
```

```
file1.txt
file2.txt
file3.txt
```

Output redirection – `stdout` and `stderr`

- Example of standard output and standard error

```
ls *.c *.txt
```

```
ls: cannot access *.c: No such file or directory   ← stderr
file1.txt   file2.txt   file3.txt                  ← stdout
```

- Redirection of standard output + standard error

```
ls *.c *.txt &> files ← redirect stdout + stderr
```

```
cat files
```

```
ls: cannot access *.c: No such file or directory
file1.txt
file2.txt
file3.txt
```

Appending output

- When redirecting output to a file, the specified file is either created (if it does not exist), or overwritten (if it exists)
- The '>>' operator tells the shell to append the output to the file, without overwriting it

Appending output – example

```
echo first line > lines.txt
echo second line >> lines.txt
echo third line >> lines.txt
cat lines.txt
```

```
first line
second line
third line
```

```
echo fourth line > lines.txt
cat lines.txt
```

```
fourth line
```

Input redirection

- We use the '<' operator to redirect the contents of a file into the input of a command
- This is useful for automation of commands that typically require typing the input by the keyboard
- Input redirection with '<' is not very common because most Linux commands do not require input from the keyboard, and for those who do, we can also use pipes (…later today)

Class assignment 2.2

Reminder

`head -nN` – show first *N* lines
`head -n-N` – show all but last *N* lines
`tail -nN` – show last *N* lines
`tail -n+N` – show all lines starting from line *N*

- Copy last 10 lines of `phone_dir.txt` into `file1.txt`
- Copy all but the last two lines of `phone_dir.txt` into `file2.txt`
- Count lines and words in `phone_dir.txt`
- Copy `phone_dir.txt` other than lines 4-6 into `file3.txt` (two commands)

Class assignment 2.2

Reminder

`head -nN` – show first *N* lines
`head -n-N` – show all but last *N* lines
`tail -nN` – show last *N* lines
`tail -n+N` – show all lines starting from line *N*

- Copy last 10 lines of `phone_dir.txt` into `file1.txt`

```
tail phone_dir.txt > file1.txt
```

- Copy all but the last two lines of `phone_dir.txt` into `file2.txt`

```
head -n-2 phone_dir.txt > file2.txt
```

- Count lines and words in `phone_dir.txt`

```
wc -lw phone_dir.txt
```

- Copy `phone_dir.txt` other than lines 4-6 into `file3.txt` (two commands)

```
head -n3  phone_dir.txt > file3.txt
tail -n+7 phone_dir.txt >> file3.txt
```

Lecture overview

- Text processing commands (`head, tail, wc`)
- Redirection
- Pipes ←
- Text delimited tables (`cut, paste, sort`)
- Simple character-based editing (`tr`)

Pipes

- The following commands count how many '`.txt`' files are in the current directory

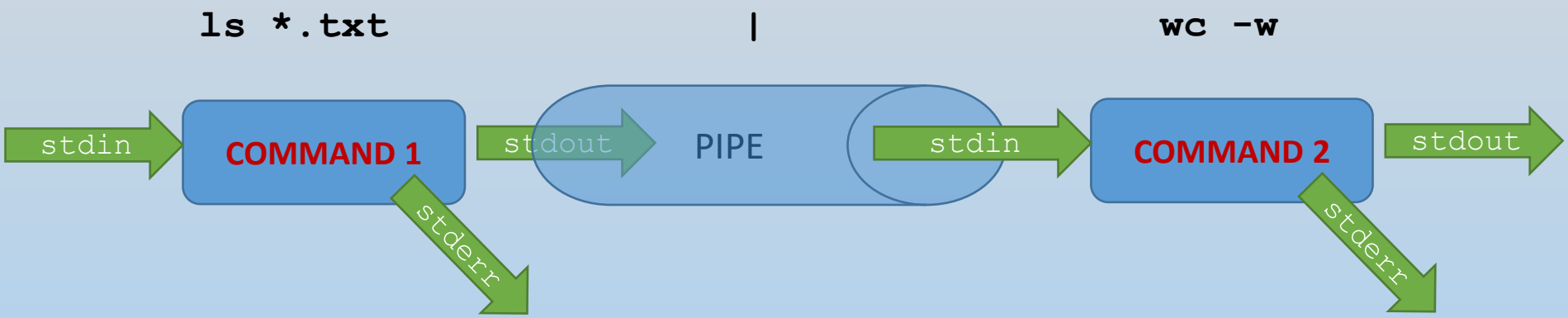
```
ls *.txt > files
wc -w files
```

  - Assuming filenames have no spaces

- This is not very efficient, and it also leaves behind a temporary file as a side effect
- What we really want is to link the output of `ls` directly to the input of `wc`

Pipes

A *pipe* takes the output of one command, and passes it as input to another



* when command **wc** reads input from stdin, it only prints the counts (no file names)

Pipes

- The two commands from the previous example can be simply combined:

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

- The symbol for a pipe is a vertical bar – '|'
- The format of a command line which uses a pipe is:

```
command1 [args] | command2 [args]
```

Pipes

- The use of *pipes* is actually a simple and easy form of inter-process communication
- Pipes can also be chained, so complex utilities can be created by combining simple building blocks
- When data passes through several pipes, these are sometimes referred to as *filters*

Pipes – examples

- Print lines 3 and 4 of '`phone_dir.txt`':

```
head -n4 phone_dir.txt | tail -n2
```

```
BAYES, Ryan 6585
BECK, Bill 6346
```

- Another way to get the same results:

```
tail -n+3 phone_dir.txt | head -n2
```

- To find the most recently changed file:

```
ls -t | head -n1
```

Sorts file in reverse order of update time

Lecture overview

- Text processing commands (`head, tail, wc`)
- Redirection
- Pipes
- Text delimited tables (`cut, paste, sort`) ←
- Simple character-based editing (`tr`)

Text delimited tables

- Each line is considered as a series of **fields** separated by **delimiter** characters
- The delimiter is set by default to `"\t"` (<tab>), but it can be defined as any other character, such as `" "` and `":"`
- You can `cut`, `paste`, and `sort` text delimited tables using UNIX commands (see 'linux-commands' guide)

Text delimited tables – examples

`phone_dir.txt`:

```
ADAMS, Andrew 7583
BARRETT, Bruce 6466
BAYES, Ryan 6585
BECK, Bill 6346
BENNETT, Peter 7456
GRAHAM, Linda 6141
HARMER, Peter 7484
MAKORTOFF, Peter 7328
MEASDAY, David 6494
NAKAMURA, Satoshi 6453
REEVE, Shirley 7391
ROSNER, David 6830
```

Cutting

The command **`cut`**: prints a subset of chars/fields of every line in a file

Usage: **`cut [OPTIONS] [FILES]`**

Common options:

- -d<C> – use character C as the field delimiter [example: `-d" "`]
- -c<RANGE> – output chars in RANGE [example: `-c3-4,6,9-`]
- -f <RANGE> – output fields in RANGE [example: `-f3-4,6,9-`]

Specify either **`-c`** or **`-f`**

Notes:

- Default field delimiter is tab (`"\t"`)
- RANGE is specified with dashes `"-"` and commas `","`
- If no files are specified, then reads text from standard input

Cutting – examples

- Print the first 3 first names from `phone_dir.txt`

```
cut -f2 -d" " phone_dir.txt | head -n3
```

```
Andrew
Bruce
Ryan
```

  - Consider space character as delimiter

- Print the last 2 last names from `phone_dir.txt`

```
cut -f1 -d"," phone_dir.txt | tail -n2
```

```
REEVE
ROSNER
```

  - Consider comma character as delimiter to separate last name from rest of the line

Pasting

The command **`paste`**: puts files side by side

Usage: **`paste [OPTIONS] [FILES]`**

Common options:

- -d<C> – use character C as the field delimiter [example: `-d" "`]

Notes:

- Default field delimiter is tab (`"\t"`)

Pasting – examples

- The file `zipcodes` contains 5-digit zipcodes
- Task I: Paste the content of `phone_dir.txt` side by side with the content of `zipcodes`; separate these 2 tables with a tab. Then show just the first three lines.

```
paste -d"\t" phone_dir.txt zipcodes | head -n3
```

```
ADAMS, Andrew 7583      34980
BARRETT, Bruce 6466     23548
BAYES, Ryan 6585        53560
```

Pasting – examples

- Task II: Multiply each line of data in `zipcodes`, by having the data on each line repeat itself 3 times (the repetitions should be separated by colons).

```
paste -d"::" zipcodes zipcodes zipcodes
```

```
34980:34980:34980
23548:23548:23548
53560:53560:53560
...
```

Sorting

The command **`sort`**:

Sorts lines of given file (or **`stdin`**) according to lex. order

Command line options:

- **`-k<N>,<M>`**
  - sort lines based on a key defined using the string that starts in the *N*-th word and ends at the end of the *M*-th word
  - words are separated by any sequence of white spaces (tab, space, etc.)
  - you can use multiple keys to define a sequence of sorting priorities
- **`-r`** – reverse order
- **`-n`** – sort based on numeric order – e.g., 9 before 21 (not lex. order, which puts 21 before 9)

Sorting

The command **`sort`**:

Sorts lines of given file (or **`stdin`**) according to lex. order

Command line options:

- **`-k<N>,<M>`**

Example:

```
cat file.txt
```

```
a 1001 3
d 1003 1
e 1004 2
b 1001 2
f 1001 1
c 1002 4
```

```
sort -k2,2 file.txt
```

```
a 1001 3
b 1001 2
f 1001 1
c 1002 4
d 1003 1
e 1004 2
```

Sorts based on 2nd field; ties are kept in original order

```
sort -k2 file.txt
```

```
f 1001 1
b 1001 2
a 1001 3
c 1002 4
d 1003 1
e 1004 2
```

Sorts based on 2nd-3rd fields combined

Removing consecutive identical lines

The command **`uniq`**: Collapses identical consecutive lines in a file. Often used on sorted files.

Example – sort directory based on family name (last name) and within that based on phone number; then remove duplicates:

```
sort -k1,1 -k3,3n phone_dir.txt | uniq
```

The -c option for the command **`uniq`**: **`uniq -c`** means: for each line of output, prepend it with the number of occurrences.

Example – to follow

File processing using pipes – examples

Task I – get unique set of first names:

File processing using pipes – examples

Task I – get unique set of first names:

- Step 1: cut second column

```
cut -f2 -d" " phone_dir.txt
```

- Step 2: sort

```
cut -f2 -d" " phone_dir.txt | sort
```

- Step 3: collapse duplicate lines (**uniq**)

```
cut -f2 -d" " phone_dir.txt | sort | uniq
```

```
Abe
Andrew
Bill
...
```

File processing using pipes – examples

Task II – number of people whose first name starts with each letter:

File processing using pipes – examples

Task II – number of people whose first name starts with each letter:

- Step 1-2: cut first character of second column

```
cut -f2 -d" " phone_dir.txt | cut -c1
```

cut first character

- Step 3-4: sort and count unique lines

```
cut -f2 -d" " phone_dir.txt | cut -c1 | sort | uniq -c
```

```
1 A
2 B
2 D
...
```

prepend with # of occurrences

File processing using pipes – examples

Task III – generate an incorrect phone directory, by concatenating each line of `phone_dir.txt` with a dash and an additional four digits. The four digits are taken from the third column of `phone_dir.txt`, sorted in decreasing numeric order. You may use one intermediate file.

File processing using pipes – examples

Task III – generate an incorrect phone directory, by concatenating each line of `phone_dir.txt` with a dash and an additional four digits. The four digits are taken from the third column of `phone_dir.txt`, sorted in decreasing numeric order. You may use one intermediate file.

- Step 1: cut 3rd column, sort and direct to `nums.txt`

```
cut -f3 -d" " phone_dir.txt | sort -rn > nums.txt
```

- Step 2: paste alongside `phone_dir.txt` (separated by dash)

```
paste -d"-" phone_dir.txt nums.txt
```

```
ADAMS, Andrew 7583-9749
BARRETT, Bruce 6466-7583
BAYES, Ryan 6585-7484
...
```

Lecture overview

- Text processing commands (`head`, `tail`, `wc`)
- Redirection
- Pipes
- Text delimited tables (`cut`, `paste`, `sort`)
- Simple character-based editing (`tr`) ←

Character substitution

- `tr` (translate) is used for simple character substitution

```
tr [options] set1 [set2]
```

- Input is read from standard input and written to standard output (no files)
- `tr` accepts two sets of character, of equal lengths, and replaces each character in the first list with the corresponding one in the second list

```
tr "abc" "ABC"
```

- converts a→A , b→B , and c→C

- For more complex substitution, use `sed` (next week)

`tr` – using ranges

- Sets may contain literal characters, such as "`D`", or character ranges, such as: "`a-z`" or "`DEFa-z`"

- Example – convert lower-case to upper-case:

```
tr "a-z" "A-Z"
```

`tr` – using ranges

- `tr` has some interpreted sequences to simplify the definition of sets:
  - `[:alpha:]` – all letters
  - `[:upper:]` – all uppercase letters
  - `[:lower:]` – all lowercase letters
  - `[:digit:]` – all digits
  - `[:alnum:]` – all letters and digits
  - `[:space:]` – all whitespace (e.g., the space, tab, and new line characters)
  - `[:punct:]` – all punctuation characters
  - `[CHAR*REPEAT]` – `REPEAT` copies of `CHAR`
  - `[CHAR*]` – copies of `CHAR` until `set1` length

tr – options

- `tr -d` – delete characters (i.e., translate to nothing)

```
==> echo 'ABabCD' | tr -d "Ab"
BaCD
==> echo 'ABabCD' | tr -d [:upper:]
ab
```

tr – options

- `tr -s` – squeeze consecutive repeats of a character

```
==> echo 'AB*****CD' | tr -s "*"
AB*CD
==> echo 'AB   CD    EF' | tr -s "[:space:]"
AB CD EF
==> echo 'ABCDEF' | tr -s " "
ABCDEF
```

- To both squeeze and translate:

```
==> echo 'AB*****CD' | tr -s "*" "-"
AB-CD
```

File processing using pipes – example

Go over the story *A Tale of Two Cities* and print the ten most frequent words in the file, with their number of appearances.

Question: how do you define a word?

One possible answer: a string of non-white-space characters between two white space characters.

```
cat a-tale-of-two-cities.txt | \          (print file)
   tr [:space:] "\n" | tr -s "\n" | \     (put words in separate lines)
   sort | \                               (sort words)
   uniq -c | \                            (count words – after sorting)
   sort -k1,1nr | \                       (sort based on counts)
   head
```

File processing using pipes – example

Go over the story *A Tale of Two Cities* and print the ten most frequent words in the file, with their number of appearances.

```
cat a-tale-of-two-cities.txt | \          (print file)
   tr [:space:] "\n" | tr -s "\n" | \     (put words in separate lines)
   sort | \                               (sort words)
   uniq -c | \                            (count words – after sorting)
   sort -k1,1nr | \                       (sort based on counts)
   head
```

Follow-up issues: (for recitation)

- Why is the "empty word" counted and how can we remove it?
- What about case-sensitivity?
- What to do with punctuation marks and digits: delete; or use as word separators?

Text parsing in Linux – recap

- Text is the standard interface in Linux
- This naturally seems inconvenient for us now that we are used to more intuitive interfaces involving graphics and touch.
- However, text interface is extremely powerful, because it allows all Linux commands to easily “interact” with one another in solving complex tasks.
- We saw how this approach can be used to write simple “one line programs” to solve relatively complex tasks.
- Next lecture we’ll see how regular expressions can be used to provide even more power to this set of tools.

At home…

- 1st homework assignment due Sunday (April 7)
- Read submission instructions carefully before preparing for submission
  - How to submit in pairs? (PARTNER file)
  - How to check solution? (check_ex script)
  - How to submit solution (submit_ex script)
  - How to request an extension?
- Review command options in the Linux command guide on Piazza (some from last lecture, some from this lecture)
  - For you to self-teach: `tr -c`
- Read `vi` tutorial and do self exercise to practice `vi` (not for submission)
- 2nd homework assignment will be published on Wednesday. You have two weeks to solve it, but don't wait to get started