Introductory Linux Tutorial for Life Sciences

Session 3: Viewing and creating files

Teacher: Marcel Martin

NATIONAL BIOINFORMATICS INFRASTRUCTURE SWEDEN

Command: cat – print file contents

- Con**cat**enates files and prints the content on the screen
- Also used to display a single file
- Usage: `cat <file>`

```
$ cat wishlistA.txt
more money

$ cat wishlistB.txt
less work

$ cat wishlistA.txt wishlistB.txt
more money
less work
```

Command: `head` – print first lines of a file

- `head` prints the first lines of a file
- Default: 10 lines
- Use `-n` option to change the number of lines

```
$ head -n 4 rosesRobertBurns.txt
O my Luve's like a red, red rose
That's newly sprung in June;
O my Luve's like the melodie
That's sweetly play'd in tune.
```

Command: `tail` – print last lines of a file

- `tail` prints the last lines of a file
- Default: 10 lines
- Use `-n` option to change the number of lines

```
$ tail -n 3 rosesRobertBurns.txt
And fare thee well, a while!
And I will come again, my Luve,
Tho' it were ten thousand mile.
```

- `tail -f` ("follow") prints the last lines of a file and **waits for more lines** (good for log files)

Command: `less` – a pager

- `cat` is not helpful when displaying long texts
- `more` shows one screenful (page) at a time
- `less` is its successor
- It can also scroll back/up
- Usage: `less <file>`

less keyboard shortcuts

↑ ↓	scroll up or down a line
Q	quit
	scroll down one page
Shift ⇑ G	go to the end of the document
10g	go to line 10
/abc	search for text 'abc'
n	find next occurrence of search text
h	show help for less

These shortcuts also work with `man` (because it uses `less`)

Exercise

1. Go to `~/data/poetry` and see what's there
2. Display the full contents of `aWhiteRoseJohnBoyleOreily.txt` (remember tab completion)
3. Show only the first four lines; the last three lines
4. Inspect `~/data/others/ATH_GO_GOSLIM.txt` with `less -M`
5. Search for "AT5G17960". On which line is it?
6. Bonus: What did the `-M` do?
   (Hint: Try to answer the previous question without using `-M`)

Exercise

1. Go to `~/data/poetry` and see what's there
2. Display the full contents of `aWhiteRoseJohnBoyleOreily.txt` (remember tab completion)
3. Show only the first four lines; the last three lines
4. Inspect `~/data/others/ATH_GO_GOSLIM.txt` with `less -M`
5. Search for "AT5G17960". On which line is it?
6. Bonus: What did the `-M` do?
   (Hint: Try to answer the previous question without using `-M`)

```
$ cd ~/data/poetroy
$ ls
$ cat aWhiteRoseJohnBoyleOreily.txt
$ head -n 4 aWhiteRoseJohnBoyleOreily.txt
$ tail -n 4 aWhiteRoseJohnBoyleOreily.txt
$ less -M ~/data/others/ATH_GO_GOSLIM.txt
/AT5G17960  # line 23456
```

`-M` shows line number at bottom of screen

Command: `touch` – Update file timestamps

- `touch` sets a file's modification time to the current time
- . . . but if the file does not exist, `touch` creates it
- So we can "abuse" it to create a new, empty file:
- `touch filename`

```
$ touch chapter1.txt
$ ls -l
-rw-rw-r-- 1 duck duck 0 Oct 10 21:00 chapter1.txt
```

Command: `touch` – Update file timestamps

- `touch` sets a file's modification time to the current time
- . . . but if the file does not exist, `touch` creates it
- So we can "abuse" it to create a new, empty file:
- `touch filename`

```
$ touch chapter1.txt
$ ls -l
-rw-rw-r-- 1 duck duck 0 Oct 10 21:00 chapter1.txt
```

- If we `touch` the file again, only the modification time changes

```
$ touch chapter1.txt
$ ls -l
-rw-rw-r-- 1 duck duck 0 Oct 10 22:30 chapter1.txt
```

Text editors

- Unix programs often work with **plain-text** files
- These are just a sequence of human-readable characters. No images, no tables, etc.
- **Text editors** can create and modify them
  - Graphical: *gedit*, *kate*, *Notepad++*, *Atom*, ...
  - Command-line: *nano*, *vi*/*vim*, *emacs*

Text editors

- Unix programs often work with **plain-text** files
- These are just a sequence of human-readable characters. No images, no tables, etc.
- **Text editors** can create and modify them
  - Graphical: *gedit*, *kate*, *Notepad++*, *Atom*, ...
  - Command-line: *nano*, *vi*/*vim*, *emacs*
- We focus on the command-line editors:
  - They can be used remotely
  - The graphical editors need less explanation

Text editors

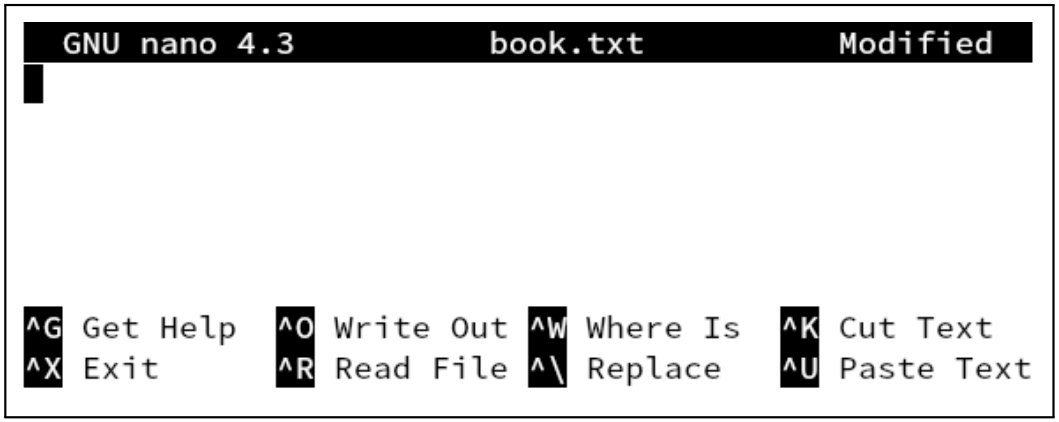
- Unix programs often work with **plain-text** files
- These are just a sequence of human-readable characters. No images, no tables, etc.
- **Text editors** can create and modify them
  - Graphical: *gedit*, *kate*, *Notepad++*, *Atom*, ...
  - Command-line: *nano*, *vi*/*vim*, *emacs*
- We focus on the command-line editors:
  - They can be used remotely
  - The graphical editors need less explanation
- Text editor $\neq$ word processor

Nano

- A command-line (keyboard-oriented) text editor

```
$ nano thesis.txt
```

- Keyboard shortcuts are shown at the bottom of the screen
- `^X` is shorthand for "Control X", `M-U` for "Alt U"



Ctrl G: Help

Ctrl O: Save file

Ctrl X: Exit nano

Ctrl W: Find text

Ctrl K/U: Cut/Paste

The vi editor

- A powerful command-line text editor
- `vi` (or `vim`) is (nearly) always available
- . . . but it requires some learning

The vi editor

- A powerful command-line text editor
- `vi` (or `vim`) is (nearly) always available
- . . . but it requires some learning
- Open (or create) a file:

```
$ vi book.txt
```

The vi editor

- A powerful command-line text editor
- `vi` (or `vim`) is (nearly) always available
- . . . but it requires some learning
- Open (or create) a file:

```
$ vi book.txt
```

```
~
~
~
~
~
~
~
~
"book.txt" 0L, 0C              0,0-1         All
```

Vi essentials

- Vi has two modes:
  - **command mode** – anything typed in this mode is interpreted as command: save file, copy and paste, find and replace, ...
  - **insert mode** – for actual typing (inserting text)
- Vi starts in *command mode*
- Press `i` to enter *insert mode*
- Press Esc to go back to command mode. Commands:
  - `:w` – save the file
  - `:x` – save and quit
  - `:q!` – quit without saving
  - `dd` – delete a line
  - `uu` – undo
- Commands starting with "`:`" require Enter

Vi essentials

- Vi has two modes:
  - **command mode** – anything typed in this mode is interpreted as command: save file, copy and paste, find and replace, ...
  - **insert mode** – for actual typing (inserting text)
- Vi starts in *command mode*
- Press `i` to enter *insert mode*
- Press Esc to go back to command mode. Commands:
  - `:w` – save the file
  - `:x` – save and quit
  - `:q!` – quit without saving ← !!!
  - `dd` – delete a line
  - `uu` – undo
- Commands starting with "`:`" require Enter

Command: wc – File statistics

- wc (word count) counts the number of lines, words, and bytes in files
- Usage: wc [options] [file]

```
$ cd ~/data/poetry

$ ls
rosesRobertBurns.txt

$ wc rosesRobertBurns.txt
19  106  527 rosesRobertBurns.txt
```

wc options

- `-l` – print the number of lines
- `-w` – print the number of words
- `-m` – print the number of characters
- `-c` – print the number of bytes
- `-L` – print the length of the longest line

```
$ wc -l rosesRobertBurns.txt
19 rosesRobertBurns.txt

$ wc -L rosesRobertBurns.txt
36 rosesRobertBurns.txt
```

Exercise

1. Open a command-line editor of your choice (hint: `nano`) for creating and editing `recipe.txt`
2. Type in your favorite recipe
3. Save the file, exit the editor
4. Find out the no. of lines, words and characters in `recipe.txt`
5. Bonus: Use `vim` to replace one ingredient with "one red cabbage"

Exercise

1. Open a command-line editor of your choice (hint: `nano`) for creating and editing `recipe.txt`
2. Type in your favorite recipe
3. Save the file, exit the editor
4. Find out the no. of lines, words and characters in `recipe.txt`
5. Bonus: Use `vim` to replace one ingredient with "one red cabbage"

```
$ nano recipe.txt
...
Ctrl-X
y
Enter
$ wc recipe.txt
```

Standard streams

- Every started program is connected by three "communication channels" to its environment
  - Standard input (stdin)
  - Standard output (stdout)
  - Standard error (stderr)

Standard streams

- Every started program is connected by three "communication channels" to its environment
  - Standard input (stdin)
  - Standard output (stdout)
  - Standard error (stderr)
- By default, these are connected as follows:
  - Standard input comes from the keyboard
  - Standard output goes to the screen
  - Standard error goes to the screen

Standard streams

- Every started program is connected by three "communication channels" to its environment
  - Standard input (stdin)
  - Standard output (stdout)
  - Standard error (stderr)
- By default, these are connected as follows:
  - Standard input comes from the keyboard
  - Standard output goes to the screen
  - Standard error goes to the screen
- They can be redirected to **files** or **other programs**
- stdout and stderr are distinct so you can redirect "normal" output (stdout), but still see error messages (stderr) on the screen

Redirecting stdout to a file

Redirect stdout to a file and *overwrite* the file if it already exists:

```
command ...  > filename
```

Redirect stdout to a file and *append to* the file if it already exists:

```
command ...  >> filename
```

Redirecting stdout to a file

Redirect stdout to a file and *overwrite* the file if it already exists:

`command ... > filename`

Redirect stdout to a file and *append to* the file if it already exists:

`command ... >> filename`

```
$ cat a_words.txt
avocado
astronomy
$ cat g_words.txt
genetics
gnu
$ head -n 1 a_words.txt > words.txt
$ cat words.txt
avocado
$ head -n 1 g_words.txt >> words.txt
$ cat words.txt
avocado
genetics
```

Redirecting stderr

- Standard error can also be redirected.
- It has *file descriptor* 2 (the "communication channel")
- Add the file descriptor before the ">":

```
command ... 2> filename
```

Standard input

- Many commands read from stdin if no filename argument is given:

```
$ wc
```

- The prompt doesn't appear because `wc` is waiting for data from stdin

Standard input

- Many commands read from stdin if no filename argument is given:

```
$ wc
```

- The prompt doesn't appear because `wc` is waiting for data from stdin
- stdin is currently the keyboard, so we type:

```
abc def
```

Standard input

- Many commands read from stdin if no filename argument is given:

```
$ wc
```

- The prompt doesn't appear because `wc` is waiting for data from stdin
- stdin is currently the keyboard, so we type:

```
abc def
```

- Then hit Enter and Ctrl D for End Of File (EOF). `wc` finishes and prints statistics (lines, words, bytes):

```
      1       2       8
```

Redirecting standard input

Run a command, but use a file as the source for stdin:

```
command ... < filename
```

```
$ cat file.txt
abc def
$ wc < file.txt
  1       2       8
```

- For `wc`, both ways work: `wc file` and `wc < file`
- But that is not the case for all commands

Pipelines

- A *pipeline* consists of multiple commands separated by the pipe symbol "|":

```
commandA [...] | commandB [...]
```

- It connects standard output of `commandA` to standard input of `commandB`

Pipelines

- A *pipeline* consists of multiple commands separated by the pipe symbol "|":

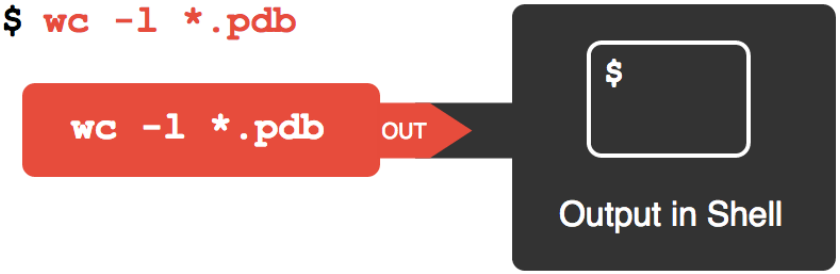
  `commandA [...] | commandB [...]`

- It connects standard output of `commandA` to standard input of `commandB`

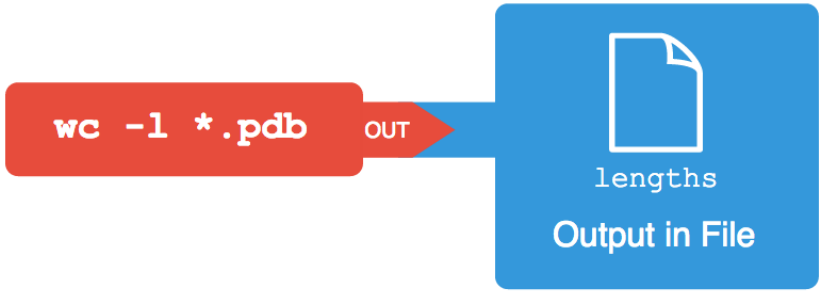
```
$ head -n 2 wonderland.txt
Alice was beginning to get very tired
of sitting by her sister on the bank,
$ head -n 2 wonderland.txt | wc
      2      15      76
```

Summary I/O redirections

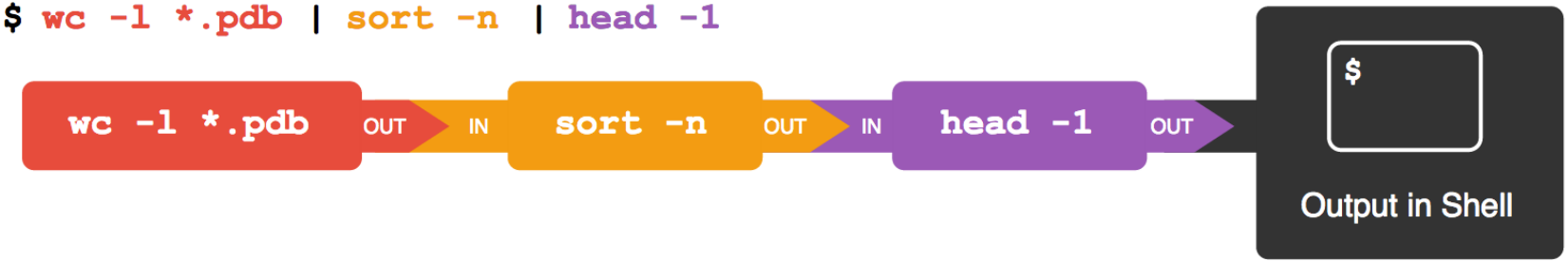
```
$ wc -l *.pdb
```




```
$ wc -l *.pdb > lengths
```




```
$ wc -l *.pdb | sort -n | head -1
```



Summary

- Display file contents with `cat`, `less`, `head`, `tail`
- Create files with `touch`
- Or use command-line editors: `nano`, `vi`, `emacs`
- Or use graphical editors: gedit, kate, Notepad++
- Get file statistics with `wc`
- Redirect standard input, standard output, standard error with `>`, `>>`, `<`, `2>`
- Build pipelines with `|`