Programming for Bioinformatics - part 3

Saul Pierotti

January 21, 2020

UNIX

- Unix was developped in 1960 at Bell labs by the founders of C
- It was one of the first OS to be multi-tasking, multi-user
- It has a hierarchical file system
- In the root directory we can find
  - `/bin` contains essential user command binaries
  - `/etc` configuration files
  - `/sbin` contains essential system binaries
  - `/usr` contains binaries and support files for user apps
  - `/var` contains variable data files
- It is written in C
- This part is about shell scripting
- Unix commands are mostly similar everywhere, but sometimes there are differences
- Alisases can be used for typing frequently used command parameters
  - They can be removed with the unalias command
  - They can be made permanent by putting in the .bashrc

Shell

- The shell is a language interpreter
- When I type a command, it searches for the command in what is in the `$PATH` variable
  - `/bin  /usr/bin  /usr/local/bin`
- In order to execute commands that are not in `$PATH`, i need to give the path
  - `./myscript`
- I can write on multiple line by putting before pressing enter

File permissions

- They work for any file (also directories, which are indeed files)
- The fundamental permissions are r, w and x and they can be applied to owner, group and all
- The combination of permission of a file are represented with 3 bits for a single user
  - 000 is no permission, 100 is r–, 010 is -w-, 001 is –x and so on
- I can express a permission status by specifying 3 numbers, and so using octal numbers
  - 0 in octal means 000 in bynary, so it is —
  - 1 means 001, so –x
  - 7 means 111. so rwx
  - 000 is ———- or d———
  - 777 is -rwxrwxrwx or drwxrwxrwx
  - 345 is –wxr–r-x

Some commands

- `echo` is the Bash way for print
- Print the working directory: `pwd`
- Create a directory: `mkdir`
- Create an empty file: `touch`
  - If I touch an existing file, I change its acces and modification time
- Copy files or directories: `cp`
  - For doing recursively (for dirs) use `cp -r`
  - It can overwrite: use `cp -i` to ask for confirmation!
    - I can also make an alias `cp=cp -i`
- Remove files: `rm`
  - There is no confirmation!
  - `rm -r` is recursive
  - `rm -i` asks for confirmation
- Remove empty directories: `rmdir`
- Move or rename: `mv`
- Scroll a file:`less`
  - I can search for words in less with `\something`
  - I can exit with `q`
  - `more` is a primitive version of `less`
- Search a file: `find`
  - I write first the directory in which I want to search and then, for instance, the name of the file
  - `find . -name myfile.txt`
  - I can also search by size, permission (`-perm`)
- Display the manual: `man`
- Path of a command: `which`
- All the paths to a command and associated files: `whereis`
- Quick one-line info on a command: `whatis`
- Info on a file: `file`
  - It tryes to guess the filetype based on its content
- Free disk space: `df`
- DIsk usage stats: `du`
- For both `df` and `du` the `-h` option makes the output human-readable
- Reverse a string: `rev`
- Simple calculations: `bc`
  - In order to operate on reals instead of integers, I should use `bc -l`

File compression

- There are many tools and hence formats
- `gzip` and `gunzip` are used for .gz files
- `tar cfz` and `tar xfz` are used for .tar
- `zip` and `unzip` are used for .zip

Network utilities

- Connect to a remote machine: `ssh`
- Copy remote files : `scp`
  - It is called secure copy
  - `scp user@remotelocation.org:path/to/file /destination/path`
- Download from the web: `wget`
  - It works with http and ftp urls

Globbing

- The Unix shell provides wildcards that can be used to specify filename patterns
  - `*` matches any number of characters, also none
    - `echo *` is equivalent to `ls`
  - `?` matches a single character
  - `[abc]` matches a, b or c
  - `[!abc]` matches not (a, b or c)
  - `[a-z]` matches any single letter
  - There are some special patterns like `[:lower:]` or `[:digit:]`
- I can specify more than 1 pattern in the same line
  - `A* T*` is equivalent to `[AT]*`
- Brace patterns can also match non-existing filenames
  - `{A,B,C}{A,B,C}` is expanded to all the 2 characters combinations of the 2 lists
  - It would be `AA AB AC BA BB BC CA CB CC`

Redirection

- In Unix devices (printers, screen output, ecc.) are treated as files
  - The `stdout` and `stderr` devices are connected to the monitor
  - `stdin` is connected to the keyboard
- `stdout` is redirected with `>`
- `stderr` is redirected with `2>`
- I can append instead of overwrite with `>>` or `2>>`
- I can redirect all the output with `&>`
  - Be careful, `&>>` does not work on all systems (!)
- The standard way to append all the output is to redirect `stderr` to `stdout` and then append it
  - I can use `ls >> file.txt 2>&1`
- I can trash an output by redirecting to `/dev/null`
- `stdin` can be redirected with `<`
  - It is almost useless, and it can not work with some commands
- Pipe (`|`) is used for redirecting the `stdout` of a command to the `stdin` of another
  - It is used for building pipelines (!)
- If I want to store an intermediate result in a pipeline, I use `tee`
  - `input command1 | tee output1.txt | command2 > output2.txt`

Text manipulation

- Concatenate and print to stdout: `cat`
  - `cat file1 file2 > file3` creates file3 containing the concatenation of file1 and file2
- Print the first/last n lines: `head -n` and `tail -n`
  - `head -4 myfile.txt` prints the first 4 lines
  - I can print a specific line by piping head and tail
    - `head -4 myfile | tail -1`
- Sort a file content: `sort`
  - The default sorting behaviour is lexicographic: 10 comes before 1
  - If I specify to sort according to a specific column I specify -k
  - columns are defined by whitespaces
  - `sort -k 2 myfile` sorts according to the second column
  - I can sort numerically with `sort -n`
  - I can remove duplicated lines with `sort -u`
    - This works only if the lines are next to each other ofter sorting (!)
- Report or omit repeated lines: `uniq`

    - It detects only adiacent duplicates (!)
        * It is algortimically complex to detect unsorted duplicates
    - `uniq -d` prints only duplicated lines
- Extract a column from a file: `cut -f`
    - `-f` specifies the field separator, that defaults to Tab
- Count stuff: `wc`
    - `wc -l` counts the lines in a file
    - `wc -c` counts the bytes (carachters including the newline)
- Compare files line by line: `diff`
    - It prints the lines that are missing from the respective files
    - It is similar to the NW algorithm
- Find common lines in sorted files: `comm`

Regex and grep

- They are more powerful than globe patterns
- `grep` means global regular expression print
    - It searches in a file and prints all the lines that match the pattern
- `egrep` can handle extended regex
- `fgrep` is fast but does not deal with regex, it searches fixed patterns
- Regex searches for a string containing the expression
    - `abc` matches ANY string containing abc
- Standard regex
    - `.` matches any single carachter
    - `*` matches any number of times (also 0) the previous carachter
    - `^b` matches anything that begins with b
    - `[ABC]` matches A, B or C
    - `^` and `$` match beginning and end
    - `A{3,5}` matches AAA, AAAA and AAAAA, `A{2,}` matches 2 or more A
    - `\` turns of a metacarachter
- Extended regex
    - `?` matches the preceding 0 or 1 time
    - `+` matches the preceding 1 or more times
    - `(exp1|exp2)` matches either exp1 or exp2
- I can invert the matching with `grep -v`
- I can search for words using `grep -w`

awk

- `awk` is a programming language designed for text processing
- The name comes from the initials of the authors
- The awk synthax can be used for text processing, arithmetic operations, string operations, and others
- It reads from STDIN or from a file
- The input is read as a set of records divided into fields
- A record is a line by default
- A field is a word by default
- The field and record separators can be changed, also to regex
- The specified operation is performed for every record, or to records that match a pattern
- The syntax is `awk 'pattern {action}' [filename]`
- If I use regex, it is `awk '/regex/ {action} [filename]`
- If action is not specified, it defaults to echo to STDOUT, essentially behaving like `grep`
- I can reference single fields of a record in the action
    - `$1`,`$2` refer to the first and second field

  - `$0` refers to the entire record
- I can specify multiple pattern with multiple actions
- There are optional blocks
  - `BEGIN{action}` and `END{action}` are executed before and after the parsing
- Variables can be built-ins or user-defined
  - Variables do not need to be declared
  - It is good practice to initialize them in the BEGIN block
- Some built-ins
  - `NR` is the number of the current record (line)
  - `NF` is the field number (word)
  - `FS` is the field separator
  - `RS` is the record separator
  - `$n` represents the nth field
- I can change the field and record separators by assigning `FS` and `RS` (!)
- Some standard operators
  - Standard arithmetic operators: '+ - * / % ++ –'
  - Relational operators: `== != < > <= >=`
  - Logical operators: `&&(AND) ||(OR) !(NOT)`
- There are 2 regex operators
  - `str ~ /regex/` returns true if str matches regex
  - `str !~ /regex/` returns true if str does not match regex
- Some string operators
  - They can change with `awk` version (!)
  - `length(str)` returns the lenght
  - `sub(regex,repl,str)` replaces the regex in str with repl
  - `substr(str,pos,len)` extracts a substring
  - `index(str,match)` returns the index of match in str, if it exists
  - `tolower(str)` and `toupper(str)` convert cases
- The print function
  - `print str1,str2` prints the strings separated by the built-in `OFS`
    - If `,` is omitted the strings are printed without separator
  - `OFS` (output field separator) can be assigned and defaults to space
  - A newline is always added at the end of the print
- Commands can be separated in a single block by `{cmd1; cmd2}`
- `getline` is used to read the next line from input
  - It is like file.readline() in python
  - Once read, the line is not read again (!)
- `printf` takes in input a format string followed by a comma-separated list of arguments
  - printf means formatted printing
  - It is similar to printf() in C, since it aek was written by the same author of C (!)
  - I can put special symbols in my string, that refer to datatypes
    - `%d %f %s` refer to int, float, string
  - I can then refer to the symbols after the print
    - `{printf "%d is an integer",1}`
  - I can remove newlines by
    - `awk '{printf "%s ",$0; getline; print $0}`
- Associatives arrays are basically dictionaries
  - `awk '{array[key]=value}'`
  - The key can also be in the form `[key1,key2,keyn]`
    - In this case it just concatenates the keys as strings and forms a complex key
  - Non-existent values default to 0 (!)
- The if statement is done like
  - `awk {if(test)codeblock;else codeblock}`
- The for loop is done like in Perl

  - `awk {for(i=0,i<10,i++) codeblock}`
- But I can also iterate on an array
  - `awk {for(key in arr) print arr[key]}`
- The while loop
  - `awk {while(test) codeblock}`