Some Programming Required

- Existing software tools can rarely do all steps
- Source data in different files may be formatted differently
- We need to bridge the differences with simple transformations
- Some programming ability is necessary for every project

Biostar: question of the day

I've been doing bioinformatics for about 10 years now. I used to joke with a friend of mine that most of our work was converting between file formats. We don't joke about that anymore.

Programming Languages

Two major groups

- Compiled languages → the output is (usually) a standalone program that can be run
- Interpreted languages → requires the presence of another software that in turn will run the software
Compiled bioinformatics code

• Compiled bioinformatics code is written mostly in C and C++

• But the code is usually composed of both
  1. the standardized language
  2. operating system specific extensions

This latter is the reason most code will not run on say the Windows operating system.

Languages that require supporting software

• Java, Perl, Python, Ruby

Very common misguided question:

Which language can be used to do X?

Turing completeness

• All languages can in principle be used to solve any computable problem.

• There is difference however in the way one solves the problem → usually there are various tradeoffs between simplicity of the solution and the speed of execution.

It is all about the libraries!

• Modern software development is all about reusing existing functionality

• Programming Languages are slowly turning into Very High Level Programming Environments → Mathematica or Matlab.
Get a good text editor

Desired features: syntax highlighting, line numbering, ability to view white space

- Komodo Edit
- Sublime Text
- Text Mate

etc. there are many other options – programmers passionately love their editors

Possibly the first scripting language: awk

- appeared in 1977, strong UNIX (1972) roots
- it is the precursor of languages such as Perl (1987) and Python (1989)
- Has fallen into disuse for a while
- Large text datasets lead to a resurgence of the language

The Art of Unix Programming: TAOUP

- Seminal open source book by Eric Raymond

For a few years after the release of Perl in 1987, awk remained competitive simply because it had a smaller, faster implementation. But as the cost of compute cycles and memory dropped, the economic reasons for favoring a special-purpose language that was relatively thrifty with both lost their force. Programmers increasingly chose to do awk-like things with Perl or (later) Python, rather than keep two different scripting languages in their heads. By the year 2000, awk had become little more than a memory for most old-school Unix hackers, and not a particularly nostalgic one.

Not quite right: The more things change the more they stay the same.

The structure of an awk program

awk ' pattern { action } '

line oriented

1. try to match the pattern to the line
2. If there is a match process the line and produce a new line
   default pattern = match everything
Simple Awk Example – no pattern specified

```bash
iabert@portkos /work
$ ls
features.gff sc.gff
iabert@portkos /work
$ awk '{ print $3 }' features.gff | head -3
telomeric_repeat
telomere
iabert@portkos /work
$ # equivalent to
telomeric_repeat
telomere
iabert@portkos /work
$ cut -f 3 features.gff | head -3
chromosome
telomeric_repeat
telomere
iabert@portkos /work
$
```

Advice
write pipelines with clear inputs and output

- Keep the flow of information from left to right
- It is a lot easier to modify/alter later on

Special variables

**Awk automatically splits** the input by whitespace (spaces and tabs) and assigns names to them:

- `$0` the entire line
- `$1` first field
- `$2` second field
- ...
- `NF` the number of fields
- `NR` the number of the current line

The whitespace curse: spaces and tabs

- Many tools will autosplit by whitespace → this was thought to be convenient but is also the source of extremely subtle errors → leads to a column shift in a tab file if a field contains spaces
- **Always specify the character to be split by!**
- This refers to programming languages as well!

Do not use the `split()` methods with their default behavior (Python, Perl etc) unless you perfectly understand what they do
Customize awk to use tabs as both the input and output field separator

```bash
alias awk="awk -F ' \t' -v OFS=' \t'"
```

Tip: you can add this to the `.profile` or `.bash_profile` file in your root folder so that it is active all the time.

Note: file names that start with a dot . are only listed if you do a `ls -a`

Recall the GFF format from lecture 3

Search for GFF → http://www.sequenceontology.org/gff3.shtml

Tab separated with 9 columns. Missing attributes may be replaced with a dot .

1. Seqid  (usually chromosome)
2. Source  (where is the data coming from)
3. Type  (usually a term from the sequence ontology)
4. Start  (interval start relative to the seqid)
5. End  (interval end relative to the seqid)
6. Score  (the score of the feature, a floating point number)
7. Strand  (+/-)
8. Phase  (used to indicate reading frame for coding sequences)
9. Attributes  (semicolon separated attributes → Name=ABC;ID=1)

Example attribute specification: `name=REB1;id=YP33546`

Make a simpler file: extract the name from 9th column -> the attribute

Semicolon separated attributes one of them is `Name=YAL069W;`

1. Split by semicolon and keep the second element, then split this again by the = sign and keep the second element

You may also put the program into a file
Write a single step at a time and test it every time

Create a gff file that contains the gene name in the type column

Make that new simpler gff: short.gff contains only genes and has name in the second column
Interval lengths for genes

Operators
- + - * / for numerical context
- % modulo division (remainder of division)
- <space> string concatenation
- ==, != equal, not equal
- ~, !~ match, no match (regular expressions)

Special patterns
- BEGIN \rightarrow before the stream starts
- END \rightarrow after the stream ends

Multiple patterns \rightarrow awk will try them each in turn
Advanced awk – usually not needed

- conditionals: if
- loops: for, while
- break, continue
- associative data structures (hash, dictionary)

You can do all that though at that point it is probably better to learn Python

But you can do a lot with just basic awk!

Awk's power comes from its simplicity – one liners!

A few awk resources

Large number of resources, the quirky name makes it very searchable → How to do X with awk?

- How to use awk – see course webpage for links

Homework 3

1. Write the awk code that finds the length of the genome (sum of all chromosome lengths) from the features.gff file.

2. Write the combination of awk and shell code that finds the longest and shortest gene in the features.gff file.