Lecture Two

Work flows and mastering your environment

Today's Topics

- FAQ from lecture one
- Actually programing
  - Breaking up tasks
  - Logical flow
  - Not going insane
- Regular expressions
  - What is a regular expression?
  - How can you use them?
- Scripting with Bash
  - Accessing and organizing programs
  - Variables, functions, and scope

FAQ

- How do I get out of man and less?
  - q

FAQ

- How do I get out of man and less?
  - q
- When using wildcard, tab completion or and other stdin, is Unix case-sensitive?
  - Yes
  - mv file1 .. will move file1
  - mv File1 .. will not move file1 and will search for File1

FAQ

- How do I get out of man and less?
  - q
- When using wildcard, tab completion or and other stdin, is Unix case-sensitive?
  - Yes
  - mv file1 .. will move file1
  - mv File1 .. will not move file1 and will search for File1
- Can Unix read spaces?
  - It reads spaces as the end of input. For example, mv My Data .. will search for a directory called 'My'

FAQ

- How can I rename a file in Unix?
  - mv filenameOld filenameNew

FAQ

- How can I rename a file in Unix?
  - mv filenameOld filenameNew

- Note on pipes and redirect
  - stdout | stdin
  - stdout >> file
  - program < stdin

FAQ

- How can I rename a file in Unix?
  - mv filenameOld filenameNew
- Note on pipes and redirect
  - stdout | stdin
  - stdout >> file
  - program < stdin
- File endings
  - They have no meaning
  - But use them for housekeeping purposes

Actually programing

- Tasks can be pretty monumental
- But if you program for hours at a time, you will *burn out.*
  - A lot of experts recommend ten minutes away from the screen for every 50 minutes at the screen

Actually programing

- But if you get up every hour, how will you remember what you were doing?

Actually programing

- But if you get up every hour, how will you remember what you were doing?
  - By breaking up tasks in a way that makes sense
  - If you stay organized from the get-go, it's easier to keep track of what you're doing

Breaking up tasks

- Most tasks can be broken up into a logical workflow
  - Ideally: develop many small tasks that you can code in an hour or a few hours

Breaking up tasks

- Most tasks can be broken up into a logical workflow
  - Ideally: develop many small tasks that you can code in an hour or a few hours
- Do I have to use different software?
  - Does this mean I need to take into account file conversions?

Breaking up tasks

- Most tasks can be broken up into a logical workflow
  - Ideally: develop many small tasks that you can code in an hour or a few hours
- Do I have to use different software?
  - Does this mean I need to take into account file conversions?
- Where can I check for errors?

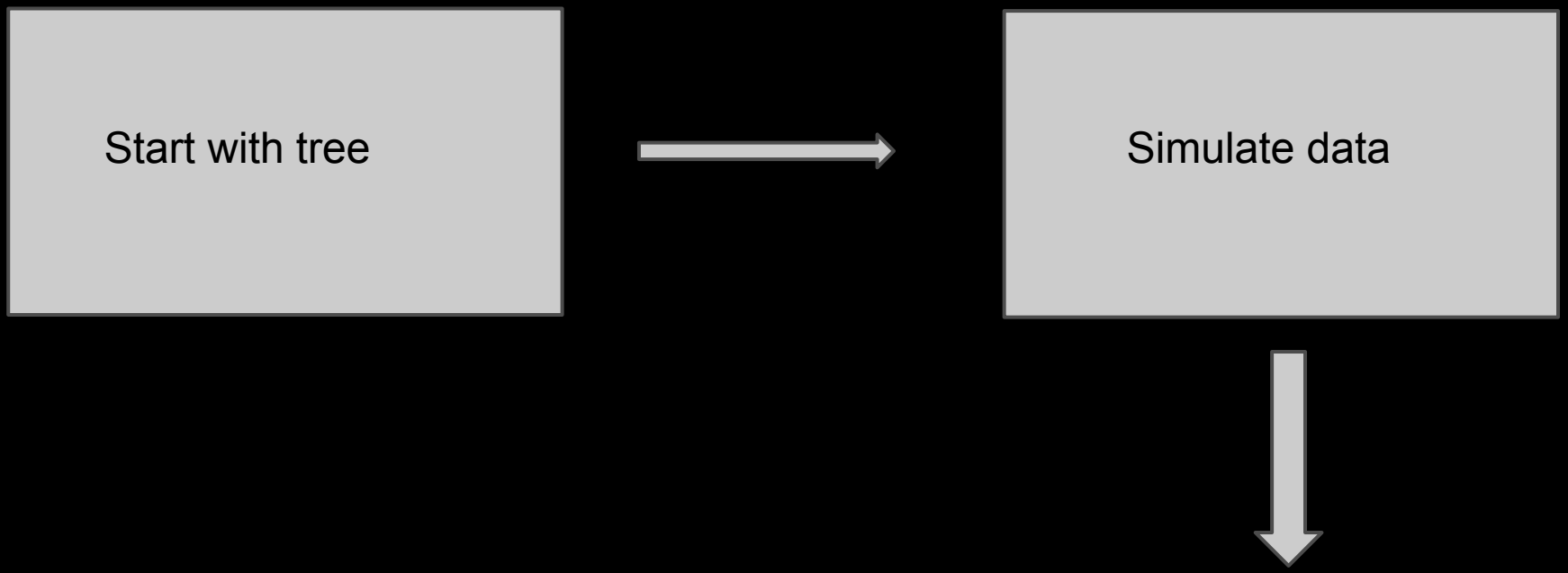
For example

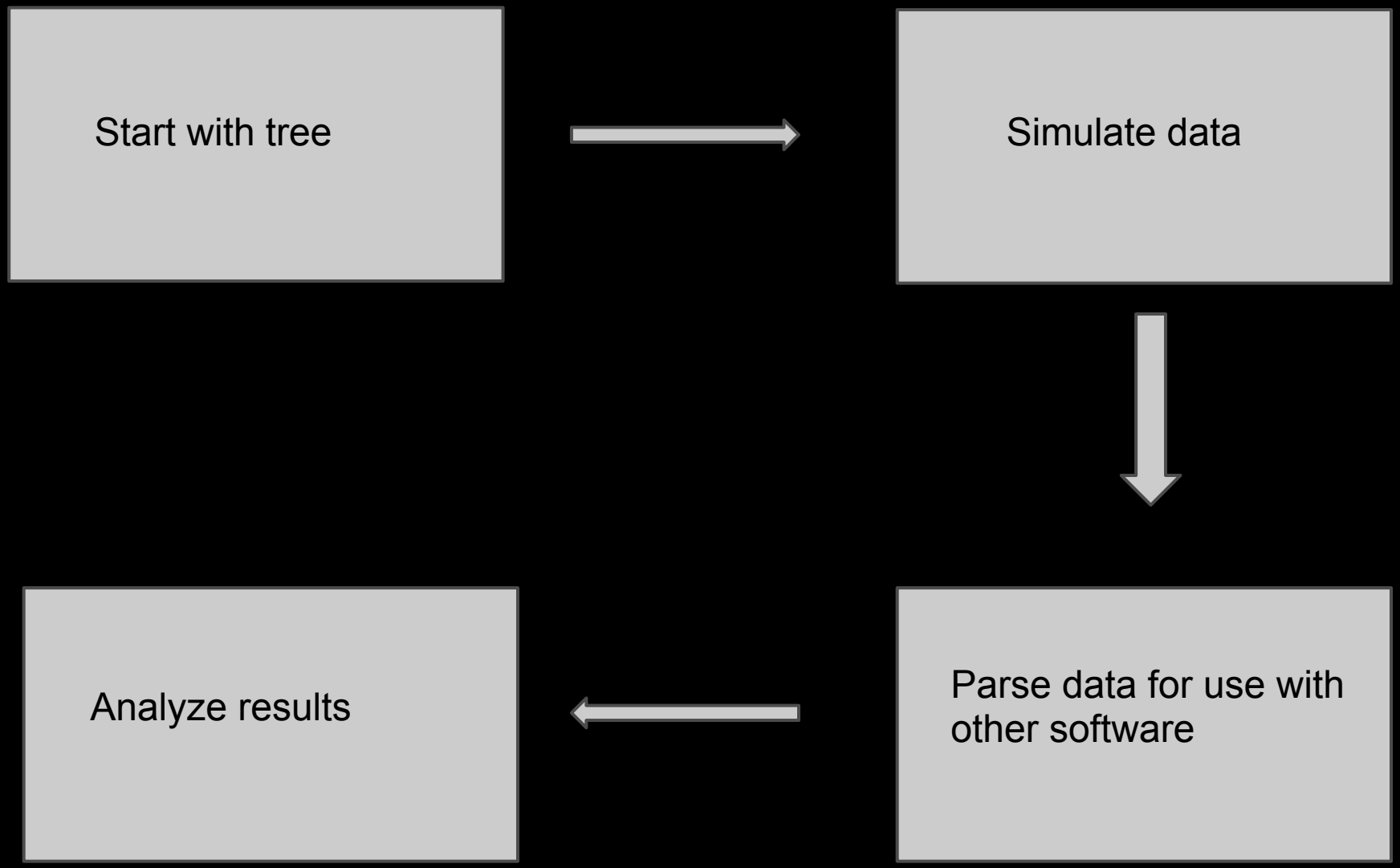
- I simulate and manipulate character matrices for phylogenetic estimation

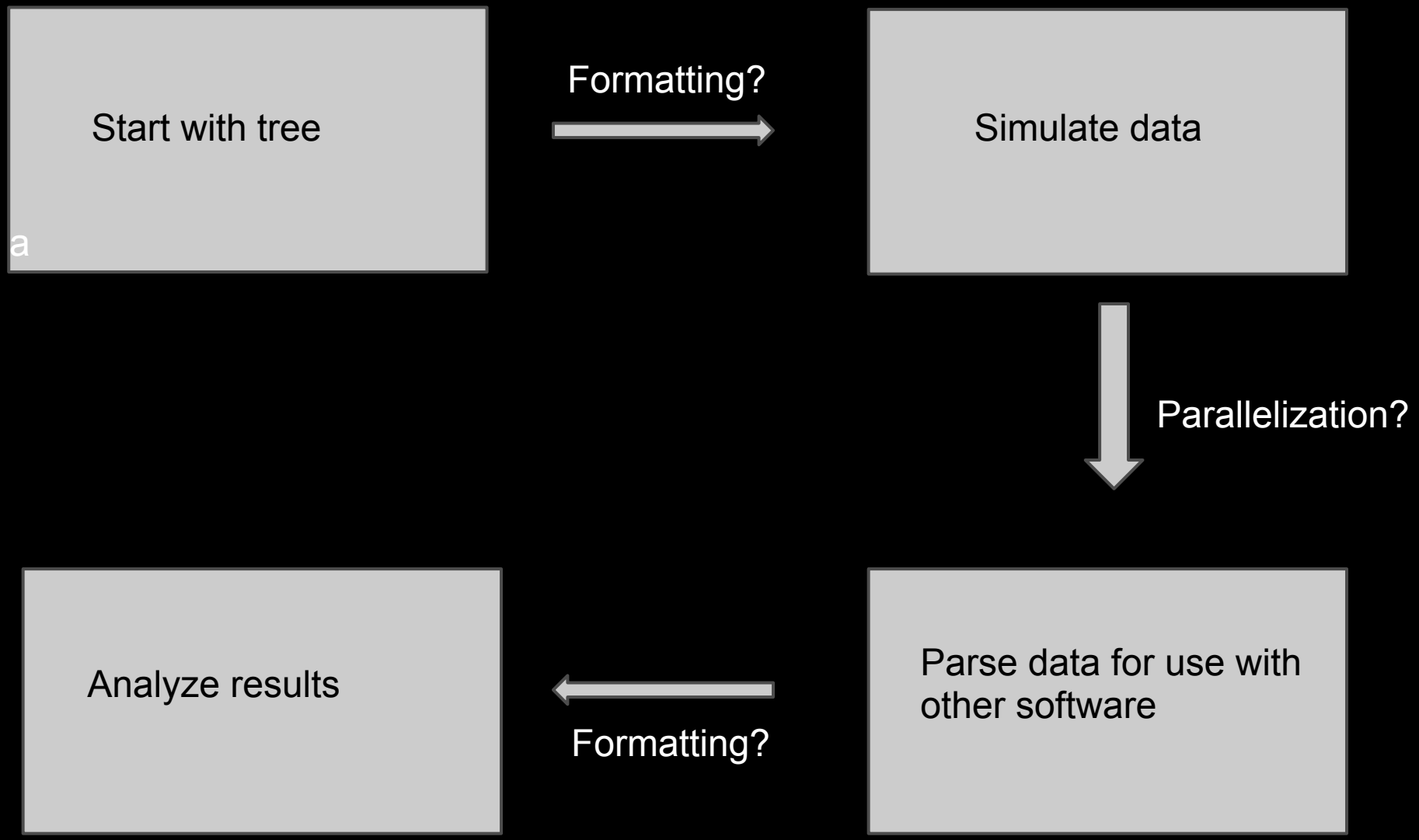
For example

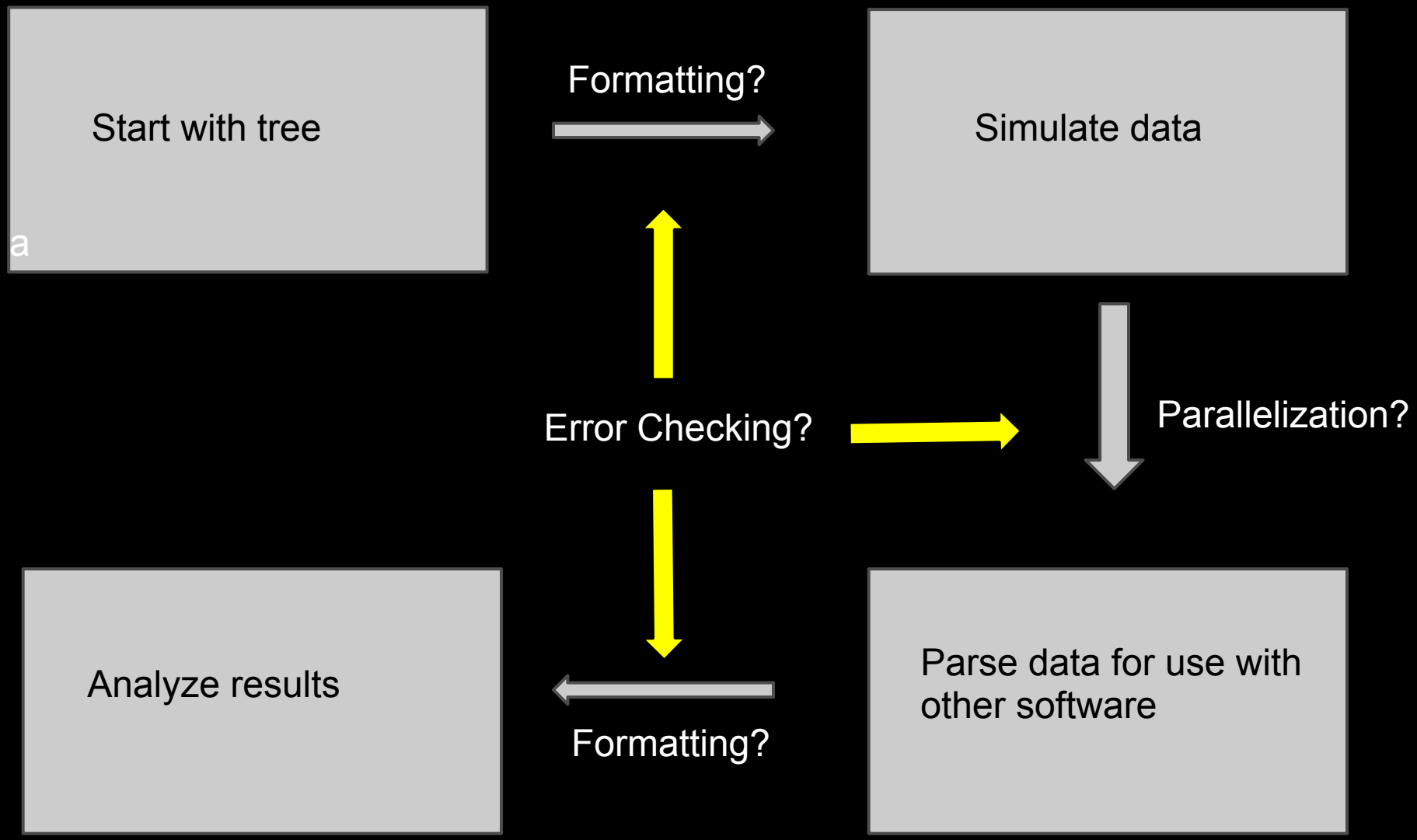
- I simulate and manipulate character matrices for phylogenetic estimation
- This is a dozen smaller tasks

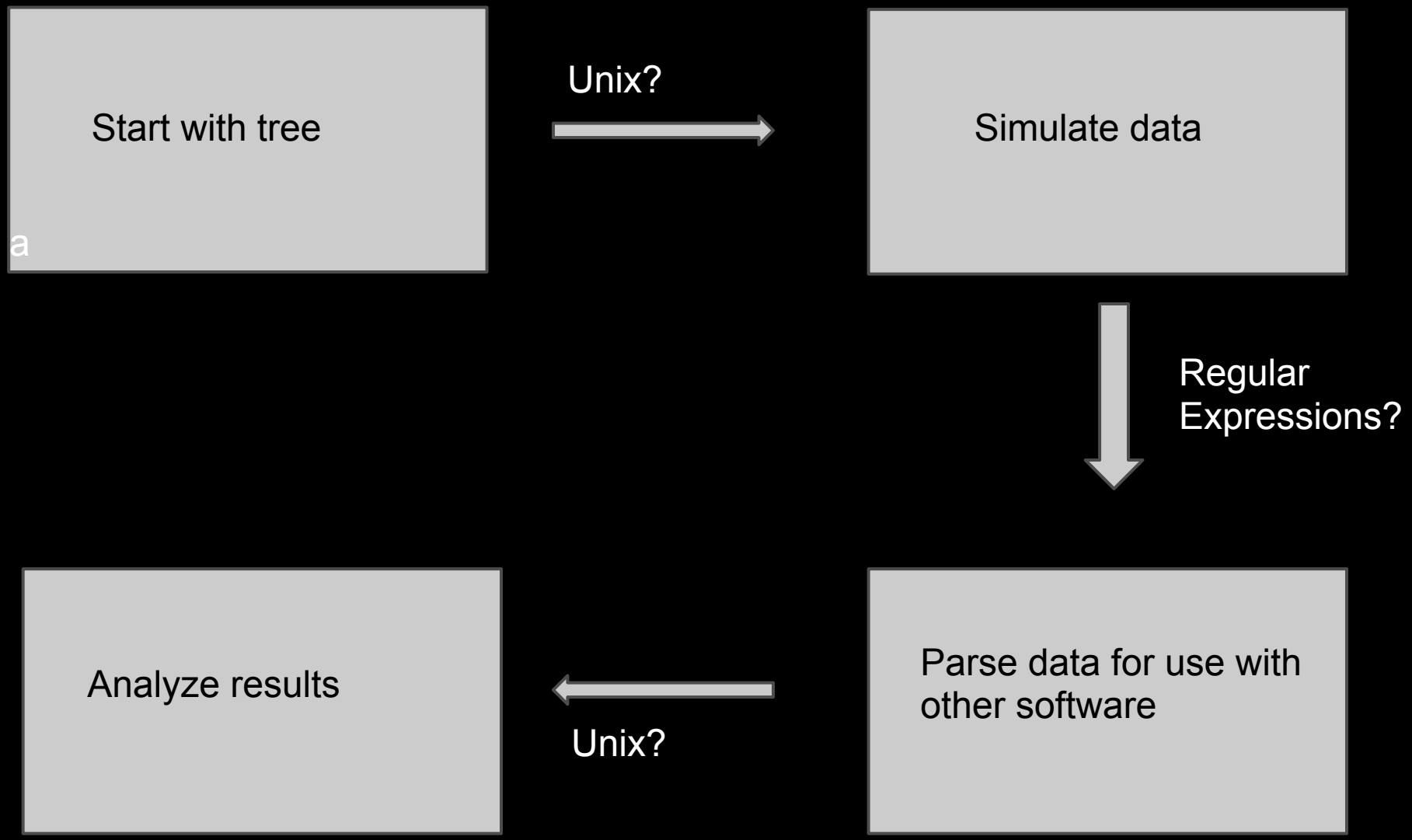
Start with tree


Start with tree
Simulate data


Start with tree
Simulate data
Parse data for use with other software
Analyze results


Start with tree
a
Formatting?
Simulate data
Parallelization?
Parse data for use with other software
Formatting?
Analyze results
Organize results?


Start with tree
a
Formatting?
Simulate data
Error Checking?
Parallelization?
Analyze results
Formatting?
Parse data for use with other software
Organize results?


Start with tree
a
Unix?
Simulate data
Regular Expressions?
Parse data for use with other software
Unix?
Analyze results
R?
Take Notes?

Regular Expressions

- Program without programming!
  - These never quite fit anywhere in a programming course
- But they're so useful!
  - Hence, we'll do them before moving any further

Regular Expression

- A flexible way of finding bits of text that match your input text

Regular Expression

- A flexible way of finding bits of text that match your input text
  - We saw this a little bit last week with grep

Regular Expression

- A flexible way of finding bits of text that match your input text
  - We saw this a little bit last week with grep

- Simplest way is to use regex in a text editor
  - Conventions may be subtly different, so watch out
  - Open search box (usually ^F)
  - You will be searching and replacing using text matching

An example

- In your DNA sequences, you have a repetitive element, AGA, and you want to find all the elements that have more than three repeats
- If you didn't know better, you might grep 'AGAAGAAGA', 'AGAAGAAGAGAGA' etc
- With regex, you can type:
  'AGA{3,}'
  And find sequences with between 3 and infinity replicates of AGA

Regex Syntax

- This can get a little hairy

Regex Syntax

- This can get a little hairy

- Regex consist of symbols compiled into little statements
  - There are a lot of possible symbols
  - http://regexpal.com/

Regex Syntax - common wildcards

Use these to search for patterns in your file

\d	Digit (0-9)
\w	Letters,numbers, and '_'
\t	Tab
\s	White space character
\r, \n, or \r\n	Line endings

Regex Symbols

- [ ]
  - The brackets specify a class of characters (such as letters, numbers or punctuation) which you are going to search
  - [a-z] will find all lowercase letters a-z
- [^]
  - Except. This is the opposite of the above. Any text after the ^ will be excluded.
- +
  - One or more of previous character
  - E.g., \w+ is one or more word characters

Regex Symbols

- \
  - The 'escape' character. This character breaks from any previous commands. This is useful because some commonly-used characters, especially punctuation have other meanings in computer language. For example, if you're looking for all characters a-c and for all periods, you could write your expression and add "\." to look for periods.
- .
  - Speaking of the period, it means any character except newline
  - \n
    - How your computer knows to go to the next line
    - \r\n on Windows (annoying!)

Regex Symbols

- ^
  - Start of the text. For example, if you wanted to find all the text that begins with F, you could type ^F
- $
  - End of the text. The same as the above, but at the end.
- |
  - Or. If you're looking for multiple patterns in the text, you might use or. For example, if I want to find Data from one of two field sites, I might use "Austin|Marfa"

Regex Syntax - Text Capture

- ()
  - Text within parentheses is stored in memory
  - In "Replace" box, to \1 or $1 to write the first captured text
- Example

  Text in: Homo sapiens

  Search: (\w)\w+ (\w\w\w)\w+

  Replace: \1\2

  Text out: Hsap

Putting it all together

- So, those are some useful symbols.
  - Use RegexPal for just a couple minutes to look at what they do.

Scripting with Unix

- We've seen how powerful command line programs are
  a. Making them programs puts you in the driver's seat

- Programs should be
  a. Easy to access
  b. Easy to edit
  c. Easy to read, and
  d. Easy to understand

- Unfortunately, none of this is easy to do

Scripting with Unix

- Easy to access
  - How to execute programs from anywhere
- Easy to edit
  - Must know where it is (should only be one copy)
  - How to organize scripts
- Easy to read
  - How to take good notes -- "commenting"
- Easy to understand
  - The task should be clear and concise
  - Write modular programs
  - Have a clear goal

Quick Word on Access

- Code should compress functional tasks
  - Code ---> Command
- This is the basis of repeatability and efficiency
- Grep example

Controlling your environment

1. Keep your house in order
   a. Make *one* scripts folder (for now)
   b. One copy of each script

2. You want to access these from anywhere
   a. Bash looks in current directory, and $PATH
   b. You could direct Bash using absolute paths:
      i. /home/ben/scripts/script1.sh file1.txt >> file2.txt
      ii. But this is lame
   c. You should edit $PATH to include your scripts folder
   d. Can also use "aliases" or "functions"

Controlling your environment

edit $PATH

1. Look at $PATH variable
   a. 'which' shows where programs live
      i. which cd, which ls, which which, which bash
   b. 'echo $PATH' shows where Bash looks for programs
2. Make a directory called scripts in your home
3. edit .bash_profile (OSX) or .bashrc (Ubuntu)
   a. Hidden files in $HOME - "ls -a" reveals them
   b. Add line at bottom:
      export PATH="$PATH:$HOME/scripts"
   c. On some Macs, the syntax may be:

```
PATH="$HOME/scripts:${PATH}"
      export PATH
```

Controlling your environment

edit $PATH

- Close terminal and reopen
  - Check with echo $PATH
- Bash now searches in ~/scripts for programs
- The syntax in the preceding commands is very important
  - If you have an error, make sure you have not inserted additional spaces
  - Make sure you don't mess with existing text - make a backup

Controlling your environment

alias and functions

- alias is a Unix command
  - alias short="really_long_command_name"
  - to permanently alias something, add to .bash_profile
- Can make functions in Unix
  - add to .bash_profile
  - funct can now be executed at command line
- More instructions on these in cheat sheet

Comments

- When you want to include information in your script that is not actual code, this is called a comment
- This is denoted by a # at the beginning of the line
- For example:

```
# This line will copy a file into a directory
cp file dir
```

Comments

- Comments make your code easier to read
  - What does this block do?
  - What's going on here?

Comments

- Comments make your code easier to read
  - What does this block do?
  - What's going on here?
  - and especially, Why?

- If you want people to use and cite your code, you need to have comments

Concepts

- Variable
  - A name and a value
  - The variable is *assigned* to a value with =
    - = is an *operator*
- Function
  - A modular block of code
  - Also called a sub-routine
  - Can be assigned to variables
- Scope
  - Where a variable can be interpreted
- Examples...

Permissions

- There's one more step
  - You've done such a good job...
  - Give yourself permission to execute your script

- chmod u+x script1.sh
  - you can now execute your script like so...
  - ./script1.sh

- With redirection...
  - ./script.sh < input_file

Exercises

- Break these big tasks into smaller ones. In a text editor, write down the steps in this task in the form of comments
- Obtain DNA sequences from GenBank and format them for an alignment program
- Format ten years of ecological data in ten different spreadsheets into one spreadsheet
- Isolate all of data points from a spreadsheet that fall beneath a threshold value