

12 - Awk/Gawk, More Git Branching

CS 2043: Unix Tools and Scripting, Spring 2016 [1]

Stephen McDowell

February 24th, 2016

Cornell University

Table of contents

1. AWK / GAWK
2. More Branching

Some Logistics

- HW2 is online...officially!

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.
 - Lecture 08 demo will be updated soon: using different separators in **sed**.

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.
 - Lecture 08 demo will be updated soon: using different separators in **sed**.
 - Excellent Piazza question: why is **read** behaving this way?

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.
 - Lecture 08 demo will be updated soon: using different separators in **sed**.
 - Excellent Piazza question: why is **read** behaving this way?
 - Directory structure sort of changed, but only in that you get more files. No changes to instructions.

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.
 - Lecture 08 demo will be updated soon: using different separators in **sed**.
 - Excellent Piazza question: why is **read** behaving this way?
 - Directory structure sort of changed, but only in that you get more files. No changes to instructions.
 - Challenge task at end.

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.
 - Lecture 08 demo will be updated soon: using different separators in **sed**.
 - Excellent Piazza question: why is **read** behaving this way?
 - Directory structure sort of changed, but only in that you get more files. No changes to instructions.
 - Challenge task at end.
- You are **FORBIDDEN** from using today's lecture in **HW2**, except for the **gandallify_extreme.sh** challenge question.

Some Logistics

- HW2 is online...officially!
- Subtle changes to **README.md**, none that are important except:
 - (OH Yesterday): I am giving sample files.
 - Lecture 08 demo will be updated soon: using different separators in **sed**.
 - Excellent Piazza question: why is **read** behaving this way?
 - Directory structure sort of changed, but only in that you get more files. No changes to instructions.
 - Challenge task at end.
 - You are FORBIDDEN from using today's lecture in **HW2**, except for the **gandalfify_extreme.sh** challenge question.
- (Poll) should I even cover Python?

AWK / GAWK

- **awk** is a programming language designed for processing text-based data.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).
 - Supports control-flow (e.g. **if-else** statements).

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).
 - Supports control-flow (e.g. **if-else** statements).
- Created at Bell Labs in the 1970s.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).
 - Supports control-flow (e.g. **if-else** statements).
- Created at Bell Labs in the 1970s.
 - Alfred Aho, Peter Weinberger, and Brian Kenrighan.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).
 - Supports control-flow (e.g. **if-else** statements).
- Created at Bell Labs in the 1970s.
 - Alfred Aho, Peter Weinberger, and Brian Kenrighan.
 - An ancestor of **perl**, a cousin of **sed**.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).
 - Supports control-flow (e.g. **if-else** statements).
- Created at Bell Labs in the 1970s.
 - Alfred Aho, Peter Weinberger, and Brian Kenrighan.
 - An ancestor of **perl**, a *cousin* of **sed**.
- Very powerful.

awk Introduction

- **awk** is a programming language designed for processing text-based data.
 - Allows easy operation on fields rather than full lines.
 - Works in a *pattern-action* manner, like **sed**.
 - Supports numerical types (and operations).
 - Supports control-flow (e.g. **if-else** statements).
- Created at Bell Labs in the 1970s.
 - Alfred Aho, Peter Weinberger, and Brian Kenrighan.
 - An ancestor of **perl**, a *cousin* of **sed**.
- Very powerful.
 - *It's Turing Complete!*

- **gawk** is the GNU implementation of the **awk** programming language.

gawk

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.

gawk

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.
- The basic structure of an **awk** program is:

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.
- The basic structure of an **awk** program is:
`pattern1 { commands }`

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.
- The basic structure of an **awk** program is:
 pattern1 { **commands** }
 pattern2 { **commands** }

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.
- The basic structure of an **awk** program is:
 pattern1 { commands }
 pattern2 { commands }
 ...

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.
- The basic structure of an **awk** program is:

```
pattern1 { commands }  
pattern2 { commands }  
...
```
- Patterns can be regular expressions!

- **gawk** is the GNU implementation of the **awk** programming language.
- On BSD/OSX, it is just called **awk**.
- On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
- **awk** allows us to setup filters to handle text as easily as numbers.
- The basic structure of an **awk** program is:

```
pattern1 { commands }  
pattern2 { commands }  
...
```
- Patterns can be regular expressions!
 - Proceeds line by line, checking each pattern one by one, executing **commands** if **pattern** is found.

Why use **awk** over **sed**?

- Convenient numerical processing.

Why use **awk** over **sed**?

- Convenient numerical processing.
- Variables and control flow in the actions.

Why use **awk** over **sed**?

- Convenient numerical processing.
- Variables and control flow in the actions.
- Convenient way of accessing fields within lines.

Why use **awk** over **sed**?

- Convenient numerical processing.
- Variables and control flow in the actions.
- Convenient way of accessing fields within lines.
- Flexible printing.

Why use **awk** over **sed**?

- Convenient numerical processing.
- Variables and control flow in the actions.
- Convenient way of accessing fields within lines.
- Flexible printing.
- Built-in arithmetic and string functions.

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```


Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

```
awk '/[Mm]onster/ {print $0}' frankenstein.txt
```

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

```
awk '/[Mm]onster/ {print $0}' frankenstein.txt
```

- The **\$0** variable in **awk** refers to the *whole line*.

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

```
awk '/[Mm]onster/ {print $0}' frankenstein.txt
```

- The **\$0** variable in **awk** refers to the *whole line*.

```
awk '/[Mm]onster/ {print $1}' frankenstein.txt
```

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

```
awk '/[Mm]onster/ {print $0}' frankenstein.txt
```

- The **\$0** variable in **awk** refers to the *whole line*.

```
awk '/[Mm]onster/ {print $1}' frankenstein.txt
```

- The first item. Can be delimited by something other than whitespace, just like **sed**.

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

```
awk '/[Mm]onster/ {print $0}' frankenstein.txt
```

- The **\$0** variable in **awk** refers to the *whole line*.

```
awk '/[Mm]onster/ {print $1}' frankenstein.txt
```

- The first item. Can be delimited by something other than whitespace, just like **sed**.

- **awk** understand extended regular expressions by default :)

Simple Examples

```
awk '/[Mm]onster/ {print}' frankenstein.txt
```

- Print all lines containing **Monster** or **monster**.

```
awk '/[Mm]onster/' frankenstein.txt
```

- If no action specified, default is to print the whole line.

```
awk '/[Mm]onster/ {print $0}' frankenstein.txt
```

- The **\$0** variable in **awk** refers to the *whole line*.

```
awk '/[Mm]onster/ {print $1}' frankenstein.txt
```

- The first item. Can be delimited by something other than whitespace, just like **sed**.
- **awk** understand **extended** regular expressions by default :)
 - We don't need to escape **+**, **?**, etc!

BEGIN and END

- **awk** allows blocks of code to be executed only once, at the beginning / end.

BEGIN and END

- **awk** allows blocks of code to be executed only once, at the beginning / end.
- With the script **monstrosity.awk** and **frankenstein.txt** in current directory:

BEGIN and END

- **awk** allows blocks of code to be executed only once, at the beginning / end.
- With the script **monstrosity.awk** and **frankenstein.txt** in current directory:

```
#!/usr/bin/awk -f
BEGIN { print "Starting search for monster..." }
/[Mm]onster/{ count++ }
END { print "Found " count " monsters in the book." }
```

BEGIN and END

- **awk** allows blocks of code to be executed only once, at the beginning / end.
- With the script **monstrosity.awk** and **frankenstein.txt** in current directory:

```
#!/usr/bin/awk -f
BEGIN { print "Starting search for monster..." }
/[Mm]onster/{ count++ }
END { print "Found " count " monsters in the book." }
```

- Use the **-f** in conjunction with shebang to cheat **awk** (it uses the script itself).

BEGIN and END

- **awk** allows blocks of code to be executed only once, at the beginning / end.
- With the script **monstrosity.awk** and **frankenstein.txt** in current directory:

```
#!/usr/bin/awk -f
BEGIN { print "Starting search for monster..." }
/[Mm]onster/{ count++ }
END { print "Found " count " monsters in the book." }
```

- Use the **-f** in conjunction with shebang to cheat **awk** (it uses the script itself).

```
>>> ./monstrosity.awk # hangs...
>>> ./monstrosity.awk frankenstein.txt # yay!
>>> awk -f monstrosity.awk frankenstein.txt # yay!
```

Important Variables

- **NF**: the number of fields in the current line.

Important Variables

- **NF**: the number of fields in the current line.
- **NR**: the number of lines read so far.

Important Variables

- **NF**: the number of fields in the current line.
- **NR**: the number of lines read so far.
 - You cannot change **NF** or **NR**.

Important Variables

- **NF**: the number of fields in the current line.
- **NR**: the number of lines read so far.
 - You cannot change **NF** or **NR**.
- **FILENAME**: the name of the input file.

Important Variables

- **NF**: the number of fields in the current line.
- **NR**: the number of lines read so far.
 - You cannot change **NF** or **NR**.
- **FILENAME**: the name of the input file.
- **FS**: the **field separator**.

Important Variables

- **NF**: the number of fields in the current line.
- **NR**: the number of lines read so far.
 - You cannot change **NF** or **NR**.
- **FILENAME**: the name of the input file.
- **FS**: the **field separator**.
 - Change **FS**="," for a **csv**.

Important Variables

- **NF**: the number of fields in the current line.
- **NR**: the number of lines read so far.
 - You cannot change **NF** or **NR**.
- **FILENAME**: the name of the input file.
- **FS**: the **field separator**.
 - Change **FS**="," for a **csv**.
 - Can also specify the **-F** flag for the **FS**.

Matching and **awk**

- **awk** can match any of the following pattern types:

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - `relational expression`

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - `relational expression`
 - `pattern && pattern`

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - `relational expression`
 - `pattern && pattern`
 - `pattern || pattern`

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - relational expression
 - `pattern && pattern`
 - `pattern || pattern`
 - `pattern1 ? pattern2: pattern3`

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - relational expression
 - `pattern && pattern`
 - `pattern || pattern`
 - `pattern1 ? pattern2: pattern3`
 - If **pattern1**, then match **pattern2**. Otherwise, match **pattern3**.

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - relational expression
 - `pattern && pattern`
 - `pattern || pattern`
 - `pattern1 ? pattern2: pattern3`
 - If `pattern1`, then match `pattern2`. Otherwise, match `pattern3`.
 - **(pattern)**: parenthesis to group / change order of operations.

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - relational expression
 - `pattern && pattern`
 - `pattern || pattern`
 - `pattern1 ? pattern2: pattern3`
 - If `pattern1`, then match `pattern2`. Otherwise, match `pattern3`.
 - `(pattern)`: parenthesis to group / change order of operations.
 - `! pattern` to invert.

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - relational expression
 - `pattern && pattern`
 - `pattern || pattern`
 - `pattern1 ? pattern2: pattern3`
 - If `pattern1`, then match `pattern2`. Otherwise, match `pattern3`.
 - `(pattern)`: parenthesis to group / change order of operations.
 - `! pattern` to invert.
 - **pattern1, pattern2**: match **pattern1**, work on every line until it matches **pattern2**.

Matching and **awk**

- **awk** can match any of the following pattern types:
 - `/regular expression/`
 - relational expression
 - `pattern && pattern`
 - `pattern || pattern`
 - `pattern1 ? pattern2: pattern3`
 - If `pattern1`, then match `pattern2`. Otherwise, match `pattern3`.
 - `(pattern)`: parenthesis to group / change order of operations.
 - `! pattern` to invert.
 - `pattern1, pattern2`: match `pattern1`, work on every line until it matches `pattern2`.
 - Cannot combine this...

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of x
- `rand()`: random number between 0 and 1

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`
- `rand()`: random number between 0 and 1
- `length(x)`: length of `x`

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`
- `rand()`: random number between 0 and 1
- `length(x)`: length of `x`
- `log(x)`: returns the log of `x`

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`
- `rand()`: random number between 0 and 1
- `length(x)`: length of `x`
- `log(x)`: returns the log of `x`
- `sin(x)`: returns the sin of `x`

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`
- `rand()`: random number between 0 and 1
- `length(x)`: length of `x`
- `log(x)`: returns the log of `x`
- `sin(x)`: returns the sin of `x`
- `int(x)`: convert to integer

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`
- `rand()`: random number between 0 and 1
- `length(x)`: length of `x`
- `log(x)`: returns the log of `x`
- `sin(x)`: returns the sin of `x`
- `int(x)`: convert to integer
- etc

Much Much More...

- Regular expression usage / comparisons:

https://www.gnu.org/software/gawk/manual/html_node/Regexp-Usage.html#index-_0021-_0028exclamation-point_0029_002c-_0021_007e-operator

- More comparison operations:

https://www.gnu.org/software/gawk/manual/html_node/Comparison-Operators.html#Comparison-Operators

- Powerful built-in functions:

- `toupper()`
- `tolower()`
- `exp(x)`: exponential of `x`
- `rand()`: random number between 0 and 1
- `length(x)`: length of `x`
- `log(x)`: returns the log of `x`
- `sin(x)`: returns the sin of `x`
- `int(x)`: convert to integer
- etc

- Wealth of information: <http://www.grymoire.com/Unix/Awk.html>

More Branching

Lecture slides...PART II!

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.

Previous cornell cs 2043 course slides.