Unix 201

Purdue University - ITaP

Gladys Andino
Dan Dietz
Jieyu Gao
Lev Gorenstein
Erik Gough
Stephen Harrell
Randy Herban
Steve Kelley
Boyu Zhang
Xiao Zhu

rcac-help@purdue.edu

February 6th and 8th, 2018

Slides available:
https://www.rcac.purdue.edu/training/unix201/

Acknowledgments

Acknowledgments

Logging In

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming

Conditionals and Loops

Acknowledgments

Acknowledgments

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

A few additional acknowledgments to the many people who have helped make this workshop possible.

- The material in this workshop was prepared by the Purdue University ITaP Research Computing team.
- Special thanks to Eric Adams and Megan Dale for organizing the workshop sessions.
- We have drawn from documentation provided by the Purdue Bioinformatics Core used in the UNIX for Biologists workshop and Next-generation Transcriptome Analysis Workshop Manual provided by Professor Michael Gribskov and Professor Esperanza Torres.

Logging In

Acknowledgments

Logging In
Windows
Mac
Activity Files

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming

Conditionals and Loops

Logging In

- Windows
- Mac
- Activity Files

Logging In

Acknowledgments
Logging In
Windows
Mac
Activity Files
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

We will be using the Radon cluster:

- www.rcac.purdue.edu/compute/radon/
- Everyone has been given an account on the cluster for the duration of the workshop
- If you wish to continue using Radon or other cluster after the workshop concludes, please make a request under your advisor or PI's name: https://www.rcac.purdue.edu/account/request/

Logging In

Windows

Acknowledgments

Logging In
Windows
Mac
Activity Files

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming

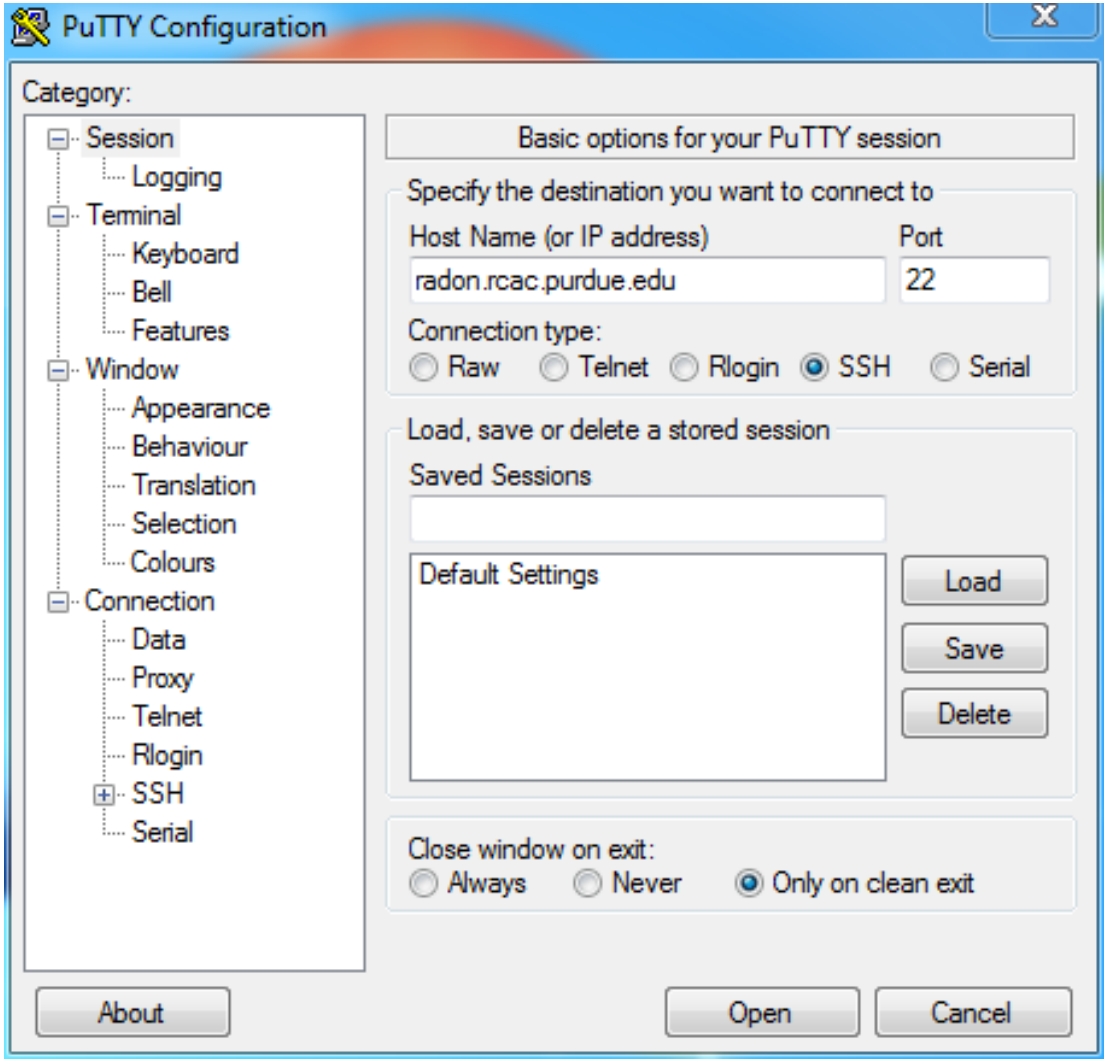
Conditionals and Loops

Many clients are available for Windows:

- We will use the PuTTY SSH client
- Download PuTTY, no install required
- http://www.chiark.greenend.org.uk/ sgtatham/putty/-download.html
  (or Google search *putty*)
- Download `putty.exe` for Intel x86 to your desktop

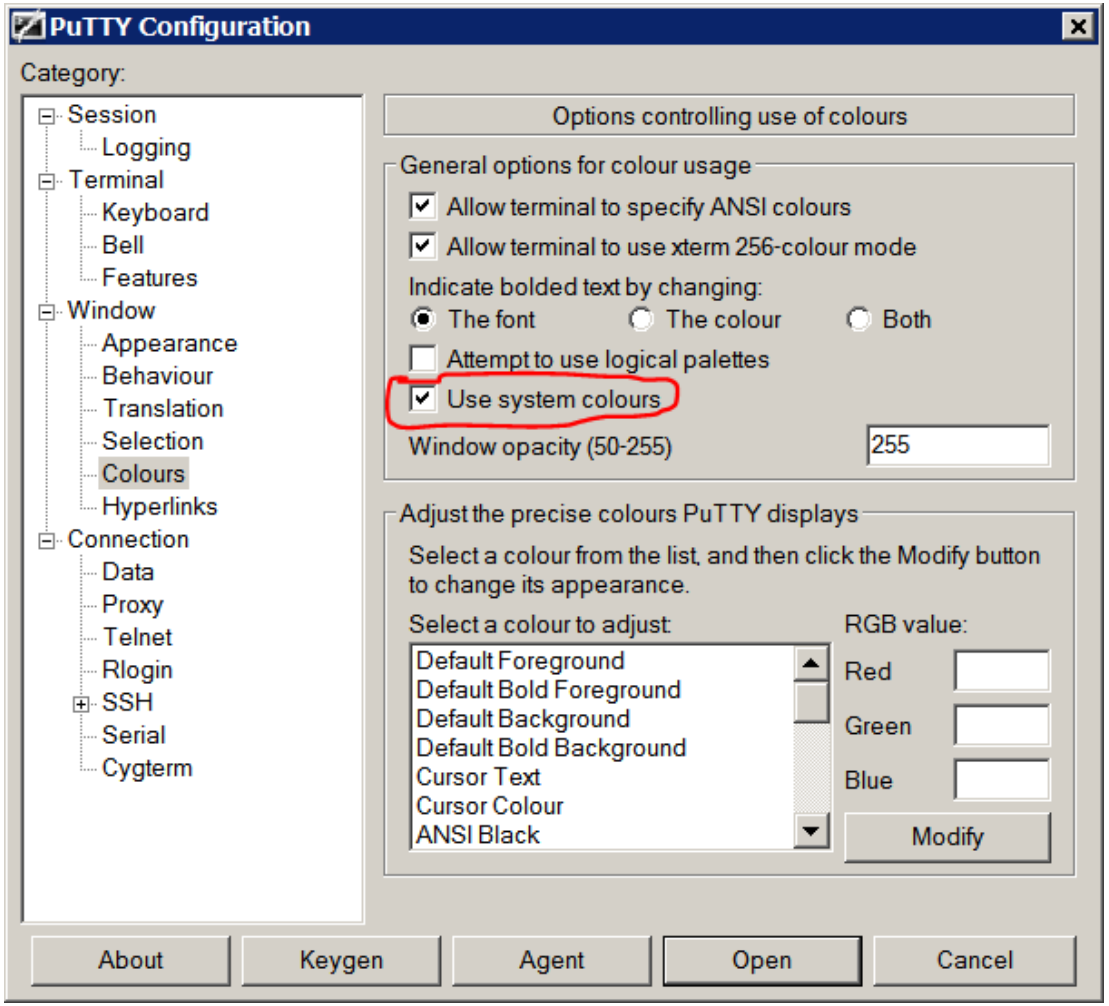
Acknowledgments
Logging In
Windows
Mac
Activity Files
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Host Name for Radon is `radon.rcac.purdue.edu`



Acknowledgments
Logging In
Windows
Mac
Activity Files
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

One tweak: enable system colors in Appearance → Colours



Logging In

Mac

Acknowledgments

Logging In
Windows
Mac
Activity Files

Text Manipulation

Regular Expressions

Advanced Text Manipulation

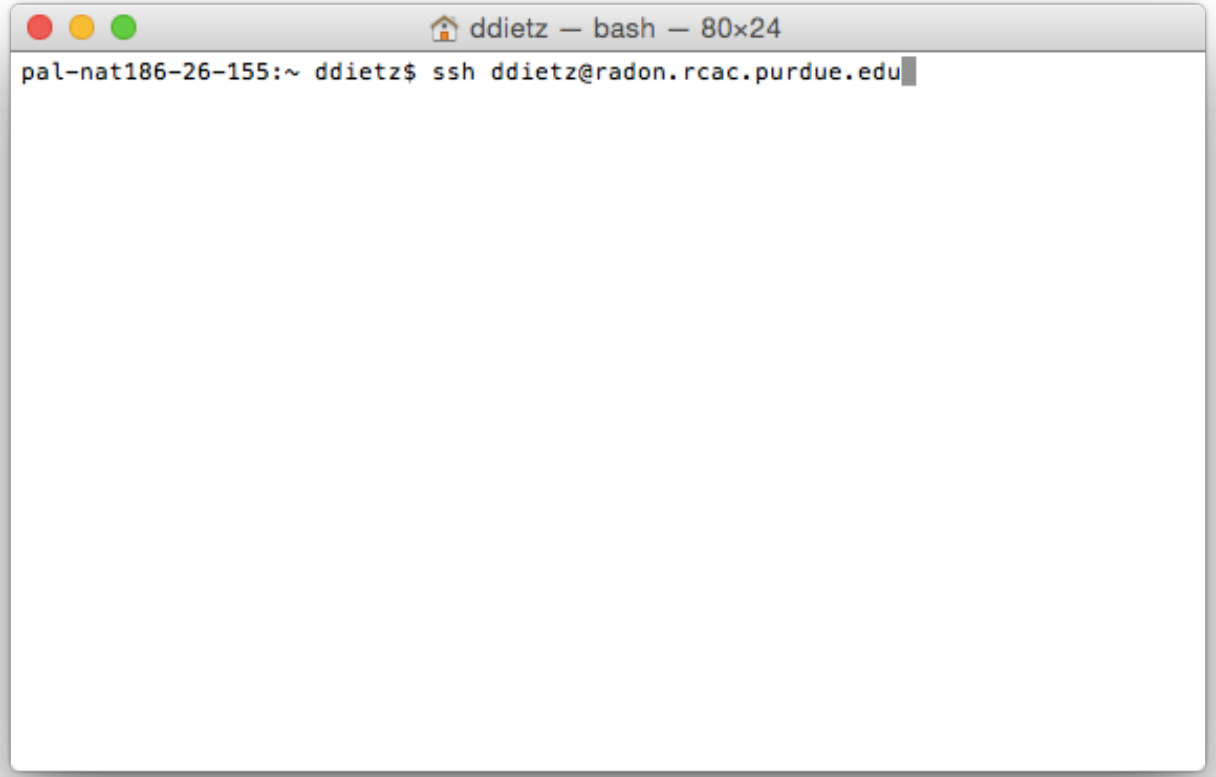
Redirects and Loops

Bash Programming

Conditionals and Loops

Connect using:

```
ssh myusername@radon.rcac.purdue.edu
```



Logging In

Mac

Acknowledgments

Logging In
Windows
Mac
Activity Files

Text Manipulation

Regular Expressions

Advanced Text Manipulation

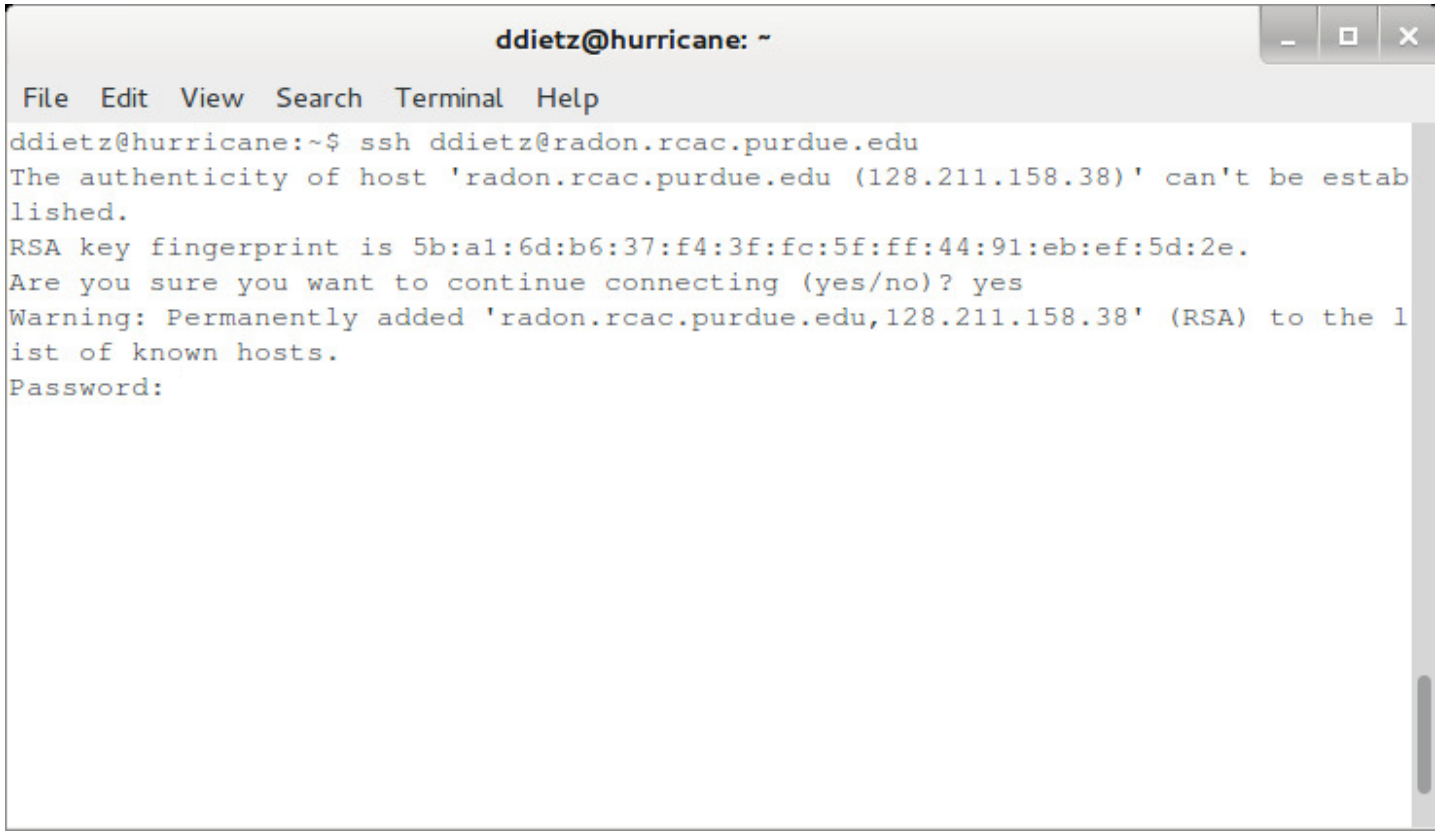
Redirects and Loops

Bash Programming

Conditionals and Loops

Linux also has a built in terminal client, similar to Mac:

```
ssh myusername@radon.rcac.purdue.edu
```



Logging In

Activity Files

Acknowledgments

Logging In
Windows
Mac
Activity Files

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming

Conditionals and Loops

We'll need a few files for some of the hands-on activities

```
$ cd
$ cp -r /depot/itap/unix101 .
```

Text Manipulation

Acknowledgments

Logging In

Text Manipulation
- wc
- cut
- sort
- uniq
- Exercises

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming

Conditionals and Loops

Text Manipulation

- wc
- cut
- sort
- uniq
- Exercises

Text Manipulation

wc

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

The `wc` (word count) command simply counts the number of lines, words, and characters.

General syntax:
`wc [OPTIONS] FILENAME`

`OPTIONS` include:

- `-l` count lines only
- `-w` count words only
- `-c` count characters only

Text Manipulation

wc

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Try this:

```
$ cd ~/unix101/Shakespeare
$ cat wcdemo.txt
This is just a very simple
text file that we'll use
to demonstrate wc

$ wc wcdemo.txt
3 14 70 wcdemo.txt
```

This tells us that the file `wcdemo.txt` has:

- 3 lines
- 14 words
- 70 characters

The we'll in the file looks like one word to Unix text processing commands.

Text Manipulation

cut

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

The `cut` command is used to select sections of each line of a file or files.

General syntax:
`cut [OPTIONS] FILENAME`

`OPTIONS` include:

- `-d` specify a character instead of TAB for field delimiter
- `-f` select only these fields; also print any line that contains no delimiter character

Text Manipulation

cut

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Try this:

```
$ cd ~/unix101/protein

$ head -n 5 1UBQ.pdb
HEADER CHROMOSOMAL PROTEIN 02-JAN-87 1UBQ
TITLE STRUCTURE OF UBIQUITIN REFINED AT 1.8 ANGSTROMS RESOLUTION
COMPND MOL_ID: 1;
COMPND 2 MOLECULE: UBIQUITIN;
COMPND 3 CHAIN: A;

$ cut -f1 -d' ' 1UBQ.pdb | head -n 5
HEADER
TITLE
COMPND
COMPND
COMPND
```

Text Manipulation

sort

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

The `sort` command is used to sort lines of a text file.

General syntax:

```
sort [OPTIONS] FILENAME
```

`OPTIONS` include:

- `-n` compare according to numerical value.
- `-r` reverse the result of comparisons.
- `-u` return only unique lines.

Notes:

- By default, lines are sorted alphabetically.
- By default, lines starting with numbers are **not** sorted numerically. For example, "8 9 10 11" would be sorted as "10 11 8 9".

Text Manipulation

sort

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Try `sort` on `words_and_num.txt`:

```
$ cd ~/unix101/Shakespeare
$ sort words_and_num.txt
```

$ sort	$ sort -n	$ sort -r
1	ana	zoo
11	MAX	MAX
23	zoo	ana
3	1	7
5	3	5
7	5	3
ana	7	23
MAX	11	11
zoo	23	1

Count unique values in first column:

```
$ cd ~/unix101/protein
$ cut -f1 -d' ' 1UBQ.pdb | sort -u | wc -l
27
```

Text Manipulation

uniq

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

The `uniq` command simply takes a sorted file and outputs the unique lines in it. The input must be sorted first.

General syntax:
`uniq [OPTIONS] INPUT`

`OPTIONS` include:

- `-c` count how many times each line occurred.
- `-d` only print duplicated lines.

Text Manipulation

uniq

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Try this:

```
$ cd ~/unix101/Shakespeare

$ sort HamletWords.txt | uniq -c | head -n 5
     36 1
     12 2
    531 a
      3 'a
      1 abate
$ sort HamletWords.txt | uniq -c | head -n 5 | sort -n
      1 abate
      3 'a
     12 2
     36 1
    531 a
```

Text Manipulation

uniq

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Try this:

```
$ cd ~/unix101/Shakespeare

$ sort HamletWords.txt | uniq -u | head -n 5
abate
abatements
abhorred
ability
Able
$ sort HamletWords.txt | uniq -u > uniques
$ cat uniques
$ sort HamletWords.txt | uniq -u | wc -l
3145
```

Text Manipulation

Exercises

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Try the following command sequence:

1. Change directory to `~/unix101/data`
2. Using a single line command, `"ls"` all the files in this directory and sort alphabetically then `"ls -l"` and sort
3. Find out how many times `"TAIR00"` and `"TAIR10"` appear in the file `at_genes.txt`
4. Using a single line command find out how many unique descriptions appear for column 3 in the `"at_genes.txt"`, please perform the search in a numerical order
5. Display 1st, 4th and 5th column of the `"at_genes.txt"` file, sorted in ascending order according to second field

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Answers:

1. Change directory to `~/unix101/data`

```
$ cd ~/unix101/data
$ pwd
/home/gandino/unix101/data
```

2. Using a single line command, "`ls`" all the files in this directory and sort alphabetically then "`ls -l`" and sort

```
$ ls | sort
at_genes.txt
awkdata.txt
grepdata.txt

$ ls -l | sort
-rw-r--r-- 1 gandino entm  215 Feb  3 18:13 awkdata.txt
-rw-r--r-- 1 gandino entm 6677 Feb  3 18:13 at_genes.txt
-rw-r--r-- 1 gandino entm  744 Feb  3 18:13 grepdata.txt
```

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Answers:

3. Find out how many times "TAIR00" and "TAIR10" appear in the file at_genes.txt

```
$ cut -f2 at_genes.txt | sort | uniq -c
 2 TAIR00
98 TAIR10

# adding grep to the line
$ cut -f2  at_genes.txt | sort |grep TAIR |uniq -c
 2 TAIR00
98 TAIR10
```

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Answers:

4. Using a single line command find out how many unique descriptions appear for column 3 in the "at_genes.txt", please perform the search in a numerical order

```
$ cut -f 3 at_genes.txt | sort | uniq -c | sort -n
1 chromosome
4 gene
5 mRNA
5 protein
6 five_prime_UTR
6 three_prime_UTR
35 CDS
38 exon
```

Acknowledgments
Logging In
Text Manipulation
wc
cut
sort
uniq
Exercises
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops

Answers:

5. Display 1st, 4th and 5th field of the "at_genes.txt" file, sorted in ascending order according to second field

```
$ cut -f1,4,5 at_genes.txt | sort -n -k 2 | head -n8
Chr1    1       30427671
Chr1    3631    3759
Chr1    3631    3913
Chr1    3631    5899
Chr1    3631    5899
Chr1    3760    3913
Chr1    3760    5630
Chr1    3996    4276
```

Regular Expressions

Acknowledgments

Logging In

Text Manipulation

Regular Expressions
- Overview
- Simple Example
- Character Groups
- Quantifiers
- Character Classes
- Escaping
- Negating
- Anchors
- Grouping
- Modifiers
- References
- Exercises

Advanced Text Manipulation

Redirects and Loops

Bash Programming

Conditionals

Regular Expressions

- Overview
- Simple Example
- Character Groups
- Quantifiers
- Character Classes
- Escaping
- Negating
- Anchors
- Grouping
- Modifiers
- References
- Exercises

Regular Expressions

Overview

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Regular expressions, what are they?

- Expression that defines a search pattern
- Can define a search for complex patterns
- Extract matches from text
- grep examples from last workshop very simple version of regular expression
- Can get way more fancy!
- Deep complex field in computer science
- Well just brush the surface and hit the basics

Regular Expressions

Overview

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

For all of these examples we will be searching the quote from Shakespeare's Hamlet:

"Though this be madness, yet there is method in it."

Yes, this is all madness but there is a reason behind it!

Regular Expressions

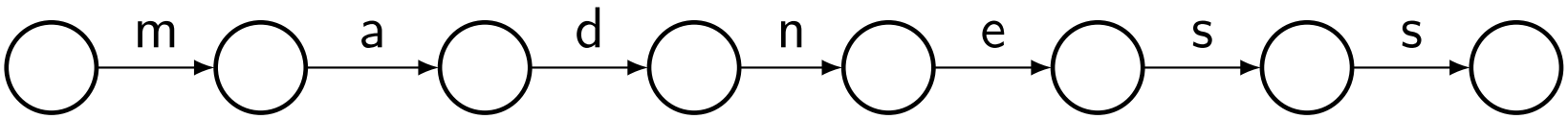
Simple Example

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Lets say we just want to search for the word "madness". Think of regular expressions as a "flow chart". Start at the beginning of the input string and expression.

Expression:
/madness/

Input:
Though this be madness, yet there is method in it.



Result:
Though this be **madness**, *yet there is method in it.*

Regular Expressions

Simple Example

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Lets try to search for the word "is"

Expression:
/is/

Input:
Though this be madness, yet there is method in it.

Result:
*Though th**is** be madness, yet there **is** method in it.*

Regular Expressions

Simple Example

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Lets try to refine this

Expression:
/ is /

Input:
Though this be madness, yet there is method in it.

Result:
Though this be madness, yet there **is** *method in it.*

Regular Expressions

Character Groups

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can define a group of characters with []

Expression:
/ madness /

Better:
/ madness[,.]/

Input:
Though this be madness, yet there is method in it.

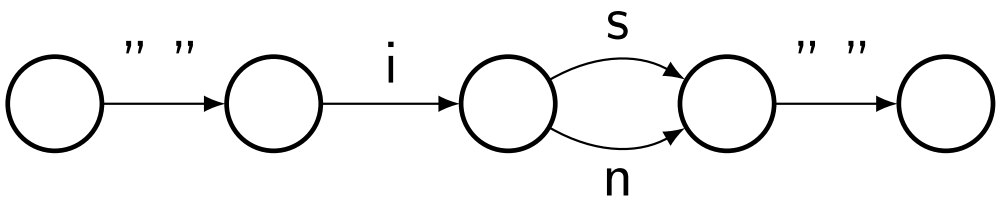
Result:
Though this be **madness,** *yet there is method in it.*

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Another example

Expression:
/ i[sn] /

Input:
Though this be madness, yet there is method in it.



Result:
Though this be madness, yet there **is** *method* **in** *it.*

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Character groups can specify range of characters:

```
[A-Za-z]
[0-9]
```

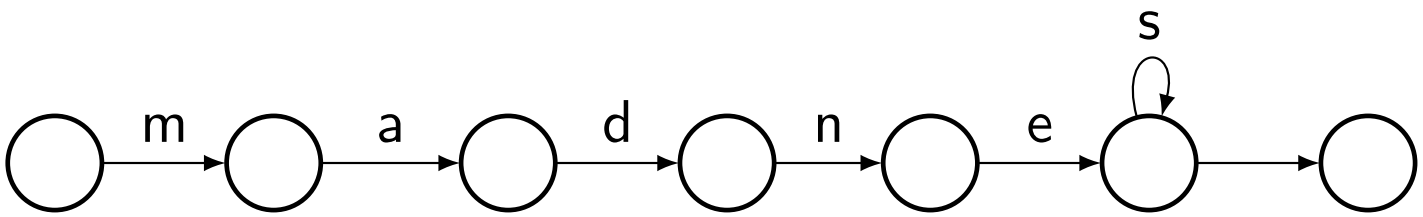
Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

We can specify *how many* of a thing we want with quantifiers:

- Use * to say "zero or more times"
- Applies to the preceding "thing" (character, group, etc)

Expression:
/madnes*/

Input:
Though this be madness, yet there is method in it.



Result:
Though this be **madness**, *yet there is method in it.*

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Another example: bogus character. Remember, **zero** or more times.

Expression:
/ madnessq*,/

Input:
Though this be madness, yet there is method in it.

Result:
Though this be **madness,** *yet there is method in it.*

Regular Expressions

Quantifiers

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can also apply to character groups

Expression:
/ madness[,.]*/

Input:
Though this be madness, yet there is method in it.

Result:
Though this be **madness,** *yet there is method in it.*

Regular Expressions

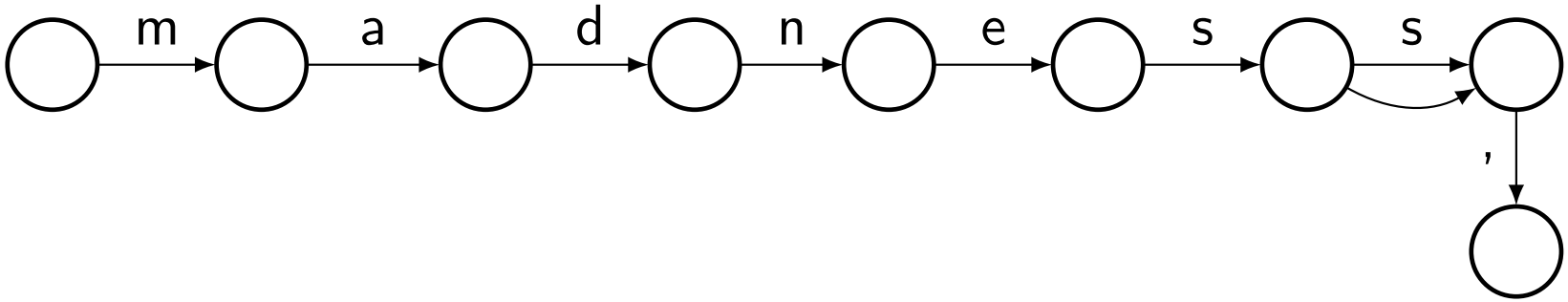
Quantifiers

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Use ? to say "zero or one times", or "optional"

Expression:
/ madness?,/
/ madnesss?,/

Input:
Though this be madness, yet there is method in it.



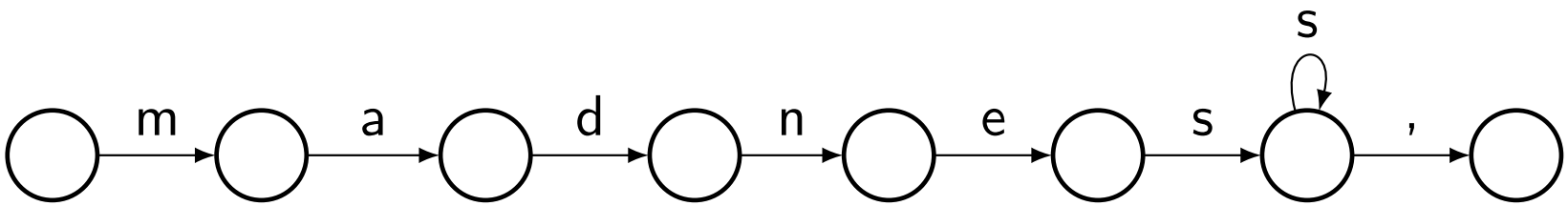
Result:
Though this be **madness,** *yet there is method in it.*

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Use + to say "one or more times"

Expression:
/ madnes+,/

Input:
Though this be madness, yet there is method in it.



Result:
Though this be **madness,** *yet there is method in it.*

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can specify precise counts with {}

Expression:
/s{2}/

Input:
Though this be madness, yet there is method in it.

Result:
*Though this be madne**ss**, yet there is method in it.*

Regular Expressions

Quantifiers

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can specify precise count ranges, or even open ended ranges

Expression:
/s{1,2}/
/s{1,}/

Input:
Though this be madness, yet there is method in it.

Result:
*Though this be madne**ss**, yet there is method in it.*

Regular Expressions

Character Classes

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Most flavors of regular expressions have the notion of a character class. They are a special syntax to specify complex character group ranges.

Word class:
`/\w+/ /[A-Za-z0-9_]+/`

Space class:
`/\s+/ /[ \t\r\n\f]+/`

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

What if we want the two words before commas?

Expression:
/\w+\s\w+,/

Input:
Though this be madness, yet there is method in it.

Result:
Though this **be madness,** *yet there is method in it.*

Regular Expressions

Character Classes

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

There is a special character ".".
It does everything!

Expression:
/madness.*/

Input:
Though this be madness, yet there is method in it.

Result:
Though this be **madness, yet there is method in it.**

Regular Expressions

Escaping

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

What if we to search for one of those special characters?
Escape with \

Expression:
/\./

Input:
Though this be madness, yet there is method in it.

Result:
Though this be madness, yet there is method in it.

Regular Expressions

Negating

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

What if we **don't** want to match something? Use ^ in a character class

Expression:
/[^mad]/

Input:
Though this be madness, yet there is method in it.

Result:
Though this be *mad***ness, yet there is** *m***etho***d* **in it.**

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

We can anchor an expression in a particular part of a string
^ for beginning of line (not to be confused with negation)

Expression:
/^\w+/

Input:
Though this be madness, yet there is method in it.

Result:
Though *this be madness, yet there is method in it.*

Regular Expressions

Anchors

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

$ for end of line

Expression:
/[\w]+[,\.!\?]+$/

Input:
Though this be madness, yet there is method in it.

Result:
Though this be madness, yet there is method in **it.**

Regular Expressions

Anchors

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can anchor at word boundaries with \b

Expression:
/\b\w+\b/

Input:
Though this be madness, yet there is method in it.

Result:
Though **this** **be** **madness**, **yet** **there** **is** **method** **in** **it**.

Regular Expressions

Grouping

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can create match groups with ()
Use | for logical or

Expression:
/\b(is|in|it|be)\b/

Input:
Though this be madness, yet there is method in it.

Result:
Though this **be** *madness, yet there* **is** *method* **in** **it**.

Regular Expressions

Grouping

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Can use quantifiers on groups

Expression:
/(\w+\s?)+/

Input:
Though this be madness, yet there is method in it.

Result:
Though this be madness, **yet there is method in it**.

Regular Expressions

Modifiers

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

There are several modifiers that can be applied to regular expressions:

- A single letter is specified after the expression
- Vary a bit from implementation to implementation, but some common ones:
  - g (global: returns ALL matches, implied on the previous examples)
  - i (case insensitive: shortcut for specify both cases)
  - m (multi-line: the ^ and $ anchor will match newlines - ie, enter key)
- Several other modifiers related to multi-line handling

Examples:
/mad/g
/mad/i

Regular Expressions

References

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

www.regular-expressions.info - Great resource for reference and everything you need to know about regular expressions.

www.regex101.com - Great tool for testing your regular expressions in various different environments.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Lets try a few of these, using the live web regex tester

Open the web page: www.regex101.com

Print the sample text, highlight and copy to your clipboard:

```
$ cd ~/unix101/regex/
$ cat hamlet_sample.txt
```

Regular Expressions

Exercises

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

What do these regular expression do:

1. /[Mm]ad/g
2. /mad/gi
3. /\w+/g
4. /\bmad\b/g
5. /\w+./g
6. /[Ww](hat|hy)?/g

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Craft a regular expression to find every word at the end of a sentence:

This business is well **ended**.--
My liege, and madam,--to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is **time**.
Were nothing but to waste night, day, and **time**.
Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief:--your noble son is mad:
Mad call I it; for to define true madness,
What is't but to be nothing else but **mad**?
But let that **go**.

Regular Expressions

Exercises

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Craft a regular expression to find every word at the beginning of a line, that starts with a W:

This business is well ended.--
My liege, and madam,--to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief:--your noble son is mad:
Mad call I it; for to define true madness,
What is't but to be nothing else but mad?
But let that go.

Regular Expressions

Exercises

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

Craft a regular expression to find a two letter word followed by a 3 letter word:

This business is well ended.--
My liege, and madam,--to expostulate
What majesty should be, what duty is,
Why day **is day**, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since brevity **is the** soul **of wit**,
And tediousness the limbs and outward flourishes,
I will be brief:--your noble son **is mad**:
Mad call I it; for to define true madness,
What is't but to be nothing else but mad?
But let that go.

Regular Expressions

Exercises

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

/\w+[.?]/g
/[A-Za-z]+[.?]/g

This business is well **ended**.--
My liege, and madam,--to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is **time**.
Were nothing but to waste night, day, and **time**.
Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief:--your noble son is mad:
Mad call I it; for to define true madness,
What is't but to be nothing else but **mad**?
But let that **go**.

Regular Expressions

Exercises

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

/^W\w+/gm
/(^|\n)W\w+/g

This business is well ended.--
My liege, and madam,--to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief:--your noble son is mad:
Mad call I it; for to define true madness,
What is't but to be nothing else but mad?
But let that go.

Regular Expressions

Exercises

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Overview
Simple Example
Character Groups
Quantifiers
Character Classes
Escaping
Negating
Anchors
Grouping
Modifiers
References
Exercises
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals

/\b\w{2}\s\w{3}\b/g
/\[A-Za-z]{2}\s[A-Za-z]{3}\b/g

This business is well ended.--
My liege, and madam,--to expostulate
What majesty should be, what duty is,
Why day **is day**, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since brevity **is the** soul **of wit**,
And tediousness the limbs and outward flourishes,
I will be brief:--your noble son **is mad**:
Mad call I it; for to define true madness,
What is't but to be nothing else but mad?
But let that go.

Advanced Text Manipulation

Acknowledgments

Logging In

Text Manipulation

Regular Expressions

Advanced Text Manipulation
grep
awk
sed

Redirects and Loops

Bash Programming

Conditionals and Loops

Advanced Text Manipulation

- grep
- awk
- sed

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

`grep` (globally search for regular expression and print)

General syntax:

```
grep [OPTIONS] PATTERN FILENAME
```

Typical scenarios:

- Extract specific line(s) from the simulation output
- Strip header/footer/comments lines from an input file
- Select files of interest
- Count number of occurrences of a pattern in a file

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Useful options:

Option	Meaning
`-v`	inverts the match (finds lines NOT containing pattern)
`--color`	colors the matched text for easy visualization
`-F`	interprets the pattern as literal string
`-E`	interprets the pattern as an extended regular expressions (more powerful, friendlier syntax)
`-H, -h`	print, don't print the matched filename
`-i`	ignore case for pattern matching
`-l`	lists the file names containing the pattern
`-n`	prints the line number containing the pattern
`-c`	counts the number of matches
`-w`	forces the pattern to match an entire word
`-x`	forces patterns to match the whole line

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Move to Shakespeare directory:

```
$ cd ~/unix101/Shakespeare/
```

Try these grep commands:

1. Search for the given string in a single file
   `grep Scene Hamlet.txt`
2. Check for the given string in multiple files
   `grep Scene *.txt`
3. Highlight the search
   `grep --color Scene Hamlet.txt`
4. Case insensitive search
   `grep -i Scene Hamlet.txt`
5. Count the number of matches
   `grep -c Scene Hamlet.txt`

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

More examples:

6. Show line number while displaying the output
   `grep -n Scene Hamlet.txt`
7. Display only the file names which matches the given pattern
   `grep -l Scene *.txt`
8. Search in all files recursively
   `grep -r Scene *`
9. Check for full words, not for sub-strings
   `grep -w all *.txt`
10. Invert match
    `grep -v a Hamlet.txt`

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Try these examples yourself using the `Lear.txt` file.

1. Find the lines that contain the word *Madam* and highlight the word.
2. Find the lines that contain the phrase *good sir* in all cases.
3. List the line number of the lines that contain the exact word *sleep*.
4. Count the number of the lines that do **not** contain the word *thy*.

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Regular expression examples using the system file `/usr/share/dict/words`. This is a file containing a list of dictionary words and is installed on all Linux systems.

1. Beginning of line (^) or end of line ($)

```
$ grep -w "^hall" /usr/share/dict/words
```

2. Character group ([0-9][a-z][A-Z])

```
$ grep "gr[ae]y" /usr/share/dict/words
$ grep "qa[^u]" /usr/share/dict/words
$ grep "[0-9]th" /usr/share/dict/words
$ grep "[0-9][0-9]th" /usr/share/dict/words
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

More regular expression examples:

3. Wildcards (use the "." for a single character match)

```
$ grep "U.S" /usr/share/dict/words
$ grep "U\.S" /usr/share/dict/words
```

Escaping the dot (\)

4. Quantifiers (?/*/+/{N}), grouping

```
$ egrep "^a.t$" /usr/share/dict/words
$ egrep "^a.?t$" /usr/share/dict/words
$ egrep "^a.*t$" /usr/share/dict/words
$ egrep "e{3}" /usr/share/dict/words
$ egrep "a{2,3}" /usr/share/dict/words
$ egrep "[ae]{2}" /usr/share/dict/words
```

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

grep OR

```
$ egrep "blue|green" /usr/share/dict/words
```

grep AND

```
$ grep blue /usr/share/dict/words | grep green
```

grep vs egrep

egrep is the same as `grep -E`. It interprets `PATTERN` as an extended regular expression.

Advanced Text Manipulation

grep

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

grep practice:

What would you expect to grep?

```
$ egrep "^[0-9]+-\w+$" /usr/share/dict/words
```

```
$ grep -i "^[^aeiou]" /usr/share/dict/words
```

Select all lines starting with a lower case letter and ending in upper case letter in `/usr/share/dict/words`.

Find the number of empty lines in the file `Hamlet.txt`?

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Change directories:

```
$ cd ~/unix101/data/
```

Working with the `grepdata.txt` file:

1. Print all lines that contain `CA` in either uppercase or lowercase.
2. Print all lines that contain an email address (they have an `@` in them), preceded by the line number.
3. Print all lines that do **not** contain the word *Sep.* (including the period).
4. Print all lines that contain the word *de* as a whole word.
5. Print all lines that contain a phone number with an extension (the letter `x` or `X` followed by four digits).
6. Print all lines that begin with 3 digits followed by a blank.
7. Print all lines that do not begin with a capital S.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

awk

A simple structured programming language. Powerful, yet simple and convenient enough for processing text organized in lines and columns.

Advanced Text Manipulation

awk

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Usage:

```
awk [OPTION] '/PATTERN/ ACTIONS' FILENAME
awk [OPTION] -f PROGRAMFILE FILENAME
```

- `PATTERN` - a regular expression.
- `ACTIONS` - statement(s) to be performed.
- several patterns and actions are possible in awk.
- `FILENAME` - input file.

Some special cases:

- No search pattern means "apply to all lines"
- No actions means "apply default action" (print the line)
- An explicitly empty action (`'{}'`) means "do nothing"

Advanced Text Manipulation

awk

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Steps:

1. Read a line from the file into a variable named $0.
2. Split up the fields. The first field is placed in variable $1, the second in $2, and so forth. Use -F to tell what the delimiter is. If you don't give a delimiter, then fields are delimited by whitespace (space, tab).
3. Do whatever command or commands are in the braces ({ and })
4. Lather, rinse, repeat.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Example:

```
Adams, Ansel;photographer;1902-02-20;1984-04-22
Asimov, Isaac;author;1920-01-02;1992-04-06
Janney, Allison;actress;1959-11-19
La Rue, Lash;actor;1917-06-15;1996-05-21
Sagan, Carl;astronomer/writer;1934-11-09;1996-12-20
Sharif, Omar;actor;1932-04-10
```

By default fields are separated by whitespace:

$1	$2	$3
Adams,	Ansel;photographer;1902-02-20;1984-04-22	
Asimov,	Isaac;author;1920-01-02;1992-04-06	
Janney,	Allison;actress;1959-11-19	
La	Rue,	Lash;actor;1917-06-15;1996-05-21
Sagan,	Carl;astronomer/writer;1934-11-09;1996-12-20	
Sharif,	Omar;actor;1932-04-10	

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Example:

```
Adams, Ansel;photographer;1902-02-20;1984-04-22
Asimov, Isaac;author;1920-01-02;1992-04-06
Janney, Allison;actress;1959-11-19
La Rue, Lash;actor;1917-06-15;1996-05-21
Sagan, Carl;astronomer/writer;1934-11-09;1996-12-20
Sharif, Omar;actor;1932-04-10
```

Use `-F';'` to get a smarter separation of fields:

$1	$2	$3	$4
Adams, Ansel	photographer	1902-02-20	1984-04-22
Asimov, Isaac	author	1920-01-02	1992-04-06
Janney, Allison	actress	1959-11-19	
La Rue, Lash	actor	1917-06-15	1996-05-21
Sagan, Carl	astronomer/writer	1934-11-09	1996-12-20
Sharif, Omar	actor	1932-04-10	

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Example:

```
Adams, Ansel;photographer;1902-02-20;1984-04-22
Asimov, Isaac;author;1920-01-02;1992-04-06
Janney, Allison;actress;1959-11-19
La Rue, Lash;actor;1917-06-15;1996-05-21
Sagan, Carl;astronomer/writer;1934-11-09;1996-12-20
Sharif, Omar;actor;1932-04-10
```

Simple printing

```
awk -F';' '{print $1, "was born", $3 "."}' people.txt
```

NF - containing # of the field in the current line

```
awk -F';' '{print $NF}' people.txt
awk -F';' 'NF < 4 {print $1 " is alive and was born in " $3}'
people.txt
```

NR - the row number being currently processed

```
awk -F';' 'NR < 3 {print $1}' people.txt
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Example:

```
Adams, Ansel;photographer;1902-02-20;1984-04-22
Asimov, Isaac;author;1920-01-02;1992-04-06
Janney, Allison;actress;1959-11-19
La Rue, Lash;actor;1917-06-15;1996-05-21
Sagan, Carl;astronomer/writer;1934-11-09;1996-12-20
Sharif, Omar;actor;1932-04-10
```

Matching pattern

```
awk -F';' '/Adams/{print $1, "was born", $3 "."}' people.txt
```

```
awk -F';' '/^A.*s/{print $1, "was born", $3 "."}' people.txt
```

Matching pattern in a field

```
awk -F';' '$3 ~ /193[0-9]/ {print $1, "was born", $3 "."}'
people.txt
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Awk variables

- It's a programming language, of course it has them!
- They can be used in either PATTERN or ACTION parts of the program.
- You can define your own.
- Some are predefined for you and can be used to change program behavior (and some even change dynamically with each read line).

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Awk variables

`FS`	Field Separator (default ANY WHITESPACE)
`OFS`	Output Field Separator (default SPACE)
`NF`	Number of Fields in the current input record (line)
`NR`	Number of Records (lines) in the input
`FNR`	File Number of Records (in current file as opposed to all input)
`RS`	Record Separator (default NEWLINE)
`ORS`	Output Record Separator (default NEWLINE)
`$N`	Nth field of the line where N can be any number (eg. $0 = entire line, $1 = first field, $2 = second field and so on). Expressions allowed: $(NF-3)
`IGNORECASE`	If not zero, regexp matching is case insensitive (default =0)

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Handy awk one-liners

awk 'NF>0 {print}' FILE	Deletes all blank lines (by the book)
awk 'NF>0' FILE	Deletes all blank lines (simpler)
awk 'NF' FILE	Deletes all blank lines (simplest)
awk 'NF>4' FILE	Prints all lines with more than 4 fields
awk '$NF>4' FILE	Prints all lines with value of the last field >4 (note the difference
awk 'END{print $NF}' FILE	Prints value of the last field of the last line
awk 'NR==25,NR==100' FILE	Prints lines between 25 and 100
awk 'END{print}' FILE	Prints the last line of the file
awk '$5=="abc123"' FILE	Prints lines which have 'abc123' in 5th field
awk 'BEGIN{ORS="\n\n"}; print' FILE	Double spaces the file
awk '{print $2,$1}' FILE	Prints only 2nd and 1st fields (swapping columns)
awk '{$2=""; print}' FILE	Prints the file without 2nd column
awk '/REGEX/' FILE	Prints all the lines having REGEX
awk '!/REGEX/' FILE	Prints all the lines not having the REGEX
awk '/AAA|BBB|CCC/' FILE	Prints all the lines having either AAA, BBB or CCC
awk 'length>50' FILE	Prints line having more than 50 characters
awk '/POINTA/,/POINTB/' FILE	Prints section of file between POINTA and POINTB

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Change directories and look at file:

```
$ cd ~/unix101/data/
$ cat awkdata.txt
```

1. Print every line from the file.
2. Print the fields that contain the name and salary.
3. Print the list of employees who has employee id greater than 200.
4. Print the list of employees in Technology department.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Change directories and look at file:

```
$ cd ~/unix101/data/
$ cat awkdata.txt
```

Print a report as below

```
Name      Designation      Department       Salary
Thomas    Manager          Sales            $5,000
Jason     Developer        Technology       $5,500
Sanjay    Sysadmin         Technology       $7,000
Nisha     Manager          Marketing        $9,500
Randy     DBA              Technology       $6,000
Report Generated
--------------
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

sed - (stream editor)

- Reads one or more text files line by line, makes changes according to editing script, and writes the results to standard output.
- Editing script can be defined to selectively add/delete/modify fragments of text (paragraph/lines/words/characters) as needed.
- Most commonly used to substitute ('s') text matching a pattern:

```
sed [OPTIONS] 's/REGEXP/REPLACEMENT/FLAGS' FILENAME
sed [OPTIONS] 'ANCHOR s/REGEXP/REPLACEMENT/FLAGS' FILENAME
```

(ANCHOR can be another regexp or some line numbers)

Advanced Text Manipulation

sed

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Change directories:

```
$ cd ~/unix101/Shakespeare/
```

sed Examples:

```
sed 's/SCENE/Scene/' Othello.txt
sed '33 s/SCENE/Scene/' Othello.txt
sed '/Castle/ s/SCENE/scene/' Othello.txt
```

See handout for more practical examples and links.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
grep
awk
sed
Redirects and Loops
Bash Programming
Conditionals and Loops

Exercises:

1. What is the output on your screen of this command line: `echo hi | sed -e 's/HI/HO/'`
   a. ho
   b. hi
   c. HO
   d. no output on screen
   e. HI
2. Which `sed` command finds every line that ends in the digits 123 and removes the first occurrence of the string `xyzzy` from those lines:
   a. `/[0-9][0-9][0-9]$/s/xyzzy//`
   b. `/xyzzy.*123$/123/`
   c. `/123$/s/xyzzy//`
   d. `s/^.*xyzzy.*123$/\1/`
   e. `/xyzzy/s/[0-9][0-9][0-9]\$//`

Redirects and Loops

Acknowledgments

Logging In

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops
- Redirects
- Pipes
- For Loops

Bash Programming

Conditionals and Loops

Redirects and Loops

- Redirects
- Pipes
- For Loops

Redirects and Loops

Redirects

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

With every UNIX program three standard *streams* are created

- Standard output (stdout):
  Normal output, printed to your screen
- Standard error (stderr):
  Error messages, printed to your screen
- Standard input (stdin):
  File for command to read in as input

Change directories:

```
$ cd ~/unix101/redirects/
```

Redirects and Loops

Redirects

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Using redirects and pipelines, we can redirect these streams elsewhere such as to a file or another command.

Why?

- Your code or program spams your screen with a ton of text and output. Rather than scrolling your screen for hours, we can send output to a file. With the output in a file, we can use one of the tools (or many others) we have talked about so far to search for interesting lines.
- Send output of one command to another one for further processing or refinement.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Change output:

`COMMAND > FILE`

Take output of a command and put it into `FILE`, rather than print it on your screen. **This overwrites `FILE` if it is already present, so be careful!**

Example:

```
$ ls -l > out.log
$ cat out.log
total 0
-rw-r--r-- 1 ddietz rcacsupp 16 Jan 24 13:07 file1.txt
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Change input:

```
COMMAND < FILE
```

Take contents of `FILE` and feed it into a command. Some commands, such as `tr`, cannot take a file name (like the commands we have seen so far) as an argument so you must feed it in by changing its standard input.

Example:

```
$ cat file1.txt
This is a file.
$ tr i u < file1.txt
Thus us a fule.
```

Redirects and Loops

Redirects

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Change input and output:

```
COMMAND < FILE > OTHERFILE
```

Example:

```
$ tr i u < file1.txt > out.log
$ cat out.log
Thus us a fule.
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Append to file rather than wipe out original:

```
COMMAND >> FILE
```

Example:

```
$ ls >> out.log
$ cat out.log
total 0
-rw-r--r-- 1 ddietz rcacsupp 16 Jan 24 13:07 file1.txt
total 0
-rw-r--r-- 1 ddietz rcacsupp 16 Jan 24 13:07 file1.txt
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Change standard error with 2>

```
$ ls -l notafile 2> error.log
$ cat error.log
ls: cannot access notafile: No such file or directory
```

Here `notafile` is a file that does **not** exist. This is done on purpose to force an error message so that redirection of standard error can be demonstrated. In real life, you probably aren't going to have errors on purpose, but should they occur you may want the error messages saved into a separate file.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Let's combine them:

```
$ ls notafile file1.txt >& out.log
$ cat out.log
ls: cannot access notafile: No such file or directory
file1.txt
```

We force an error by purposely requesting a non-existent file in addition to standard output with a real file. This generates two separate streams that we can direct into a single file (instead of printing both to your screen).

If your program generates a ton of output, it may be helpful to put it into a file so that is easy to search through later.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Divide and conquer:

```
$ ls notafile file1.txt 2> error.log > out.log
$ cat error.log
ls: cannot access notafile: No such file or directory
$ cat out.log
file1.txt
```

We force an error by purposely requesting a non-existent file in addition to standard output with a real file. This generates two separate streams that we can direct into two separate files (instead of printing both to your screen).

Redirects and Loops

Redirects

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

We can throw away errors with /dev/null

```
ls -l * 2>/dev/null
```

We can throw away everything too

```
ls -l * >& /dev/null
```

/dev/null is a special file on UNIX systems. Anything written is thrown away (permanently). Perhaps your program generates a ton of useless output. You could send the standard output into the garbage, while keeping only the error messages.

Redirects and Loops

Pipes

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

We can tell one to go into the same place as another:

```
ls -l notafile file1.txt 2>&1 |less
```

Pipes will only send standard output into the next program. Normally any messages to standard error will be printed on your screen. By combining error into out, we can pipe error messages into the next program instead of your screen.

Redirects and Loops

Pipes

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

What if we have a chatty program, want to save the output in a file for later viewing, but also want to monitor the progress of the command in real-time? A special command called `tee` can accomplish this.

```
$ ls notafile file1.txt 2>&1 | tee out.log
ls: cannot access notafile: No such file or directory
file1.txt
$ cat out.log
ls: cannot access notafile: No such file or directory
file1.txt
```

Redirects and Loops

For Loops

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

We'll discuss for loops more in bash programming, but they are useful even on the command line

```
$ for i in "one" "two" "three"; do echo $i; done
one
two
three
```

Redirects and Loops

For Loops

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Redirects
Pipes
For Loops
Bash Programming
Conditionals and Loops

Command substitution (we'll discuss more later on):

```
$ cd ~/unix101/redirects/
$ mkdir backup
$ ls *.*
error.log  file1.txt  out.log
$ for i in `ls *.*`; do cp "$i" backup/; done
```

This example takes each item from `ls *.*`, and runs a command(s) on each file. Here we are copying each file into the backup directory. Of course, this is very simplistic (you could just do `cp *.* backup/` but imagine you want to do more complex operations on a list of files, and you don't want to type the same command a bunch of times.

Be very cautious of files with spaces in the name (don't do it!) as `for` iterates by spaces (remember `awk`).

Bash Programming

Acknowledgments

Logging In

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming
- Shell basics
- Shell Types
- Variables
- String Operations
- Arithmetic Operations
- Command Substitution
- Quoting Characters

Conditionals and Loops

Bash Programming

- Shell basics
- Shell Types
- Variables
- String Operations
- Arithmetic Operations
- Command Substitution
- Quoting Characters

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

The first line of the shell script defines the program that interprets the script

```
#!/bin/bash
```

End of a command using ; or a newline

```
#!/bin/bash
ls; pwd;
cd $HOME
ls
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Make the script `FILENAME` executable

```
$ chmod +x FILENAME
```

Execute a shell script `script.sh` in dir `/path/to`

```
$ /path/to/script.sh
```

Execute a shell script script.sh in your current working directory

```
$ ./script.sh
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise:

- Change to directory:

```
$ cd ~/unix101/scripts
```

- Check the permission of the script

```
$ ls -l script1
$ -rwxr-xr-x 1 gandino student 72 Feb 15  2016 script1
# should have x!
```

- Try different ways to run the scripts

```
$ $HOME/unix101/scripts/script1
$ ./script1
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

UNIX/Linux systems offer a variety of shell types

- bash (Bourne Again shell)
- csh or C Shell
- tcsh or TENEX C Shell
- sh or Bourne Shell

Note: different shells have different syntax to do the same thing!

```
$ echo $SHELL
/usr/local/bin/bash
```

Bash Programming

Variables

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Variables may be of different value types:

- Bash does not force variables to have types
- Operations on variables depend on the content of the variables
- Depending on contents, variables are of 4 types:
  - String
  - Integer
  - Constant
  - Array

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Create variable named `VARNAME` and set equal to `value`, and then print back the value of the variable:

```
$ VARNAME="value"
$ echo $VARNAME
```

Dereference the variable `VARNAME` by placing a $ in front

- No spaces around the = sign
- Variable names
  - Case sensitive
  - A combination of letters, numbers, and underscores; names starting with numbers are invalid
  - Avoid using reserved words: if, else, fi, for
  - Avoid using environment variables: PATH, SHELL ... (see `printenv` command)

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Variables in Bash have scope, or variables in Bash are only accessible from specific environments:

- The variable created in your shell is only available to the current shell (the one you are typing in, and only the one you are typing in)
- Child processes of the current shell (such as a script you are trying execute) will not see this variable
- To pass variables to subshells or scripts, we need to export variables:

```
$ export VARNAME="value"
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercises:

- Check your current shell type and make sure it is bash

```
$ echo $SHELL
```

- Create a integer variable

```
$ INT1=765
$ echo $INT1
```

- Create a string variable

```
$ STR1="Hello World"
$ echo $STR1
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Arrays allow you to store a list of values inside a single variable:

- An array variable contains multiple values, index starts from 0
- Array declaration

```
$ declare -a MYARRAY
$ MYARRAY=(value1 value2)
```

- `declare -a MYARRAY` declares `MYARRAY` as an array variable, with no initial values
- `MYARRAY=(value1 value2)` assigns values to the array

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Once defined, elements of an array can be accessed in several ways:

- Array elements reference

```
$ MYARRAY=(value1 value2 value3)
$ echo ${MYARRAY[*]}
$ echo ${MYARRAY[0]}
$ echo ${MYARRAY}
```

- `${MYARRAY[*]}` refers to the whole array MYARRAY
- `${MYARRAY[0]}` refers to the first element of MYARRAY
- `${MYARRAY}` also refers to the first element of MYARRAY

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Individual elements of an array can be redefined at any time:

- Assign value to an array element

```
$ MYARRAY=(value1 value2 value3)
$ echo ${MYARRAY[0]}
value1
$ MYARRAY[0]=newval1
$ echo ${MYARRAY[0]}
newval1
```

- `MYARRAY[index]=val` assigns `val` to the element `MYARRAY[index]`. `val` can be of any type such as a string or number.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise; create an array:

- Name: ARRAY1
- 1st element is "hello"
- 2nd element is 10
- 3rd element is 48
- 4th element is 20
- 5th element is "world"

```
#Way 1:
$ declare -a ARRAY1
$ ARRAY1[0]="hello"
$ ARRAY1[1]=10
...
$ echo ${ARRAY1[*]}

#Way 2:
$ ARRAY1=("hello" 10 48 20 "world")
$ echo ${ARRAY1[*]}
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

There are some variables that Bash predefines for you:

- These variables can only be referenced
- `$0, $1, $2` ...: positional parameters
  - `$0`: the name of the executable as it was called
  - `$1`: first command line argument that you gave to the executable
  - `$2`: second command line argument
  - `$#`: number of command line parameters
- Example: `positional.sh`

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise:

```
$ cat positional.sh
#!/bin/bash

# positional.sh
# This script reads first 3 positional parameters
      and prints them out.

echo
echo "Name of the script being executed is $0"
echo
echo "$1 is the first positional parameter, \$1"
echo "$2 is the second positional parameter, \$2"
echo "$3 is the thrid positional parameter, \$3"
echo
echo "The total number of positional parameters is
       $#."
```

- Execute positional.sh
- Execute positional.sh: `./positional.sh hello world`
- Execute positional.sh: `./positional.sh "hello world"`
- Execute positional.sh with 3 parameters `./positional.sh "hello world 10" 20 30`
- Execute positional.sh with 5 parameters `./positional.sh hello world 10 20 30`
- Bonus: `echo $0`

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Length of a variable ${#VAR}

```
$ echo $SHELL
/usr/local/bin/bash
$ echo ${#SHELL}
19
```

String concatenation STR="$STR1$STR2"

```
$ str1="Hello"
$ str2="World"
$ str="$str1 $str2"
$ echo "$str"
Hello World
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Substring extraction ${VAR:OFFSET:LENGTH}

- OFFSET: the index of the character the substring starts from
- OFFSET starts from 0
- LENGTH: the number of characters to keep in the substring
- When LENGTH is omitted, the reminder of the string is taken

```
$ MYSTRING="thisisaverylongname"
$ echo ${MYSTRING:4}
isaverylongname

$ echo ${MYSTRING:6:5}
avery
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise; create a string variable:

- Name: STR
- Value: "Welcome to Research Computing"

Create a substring of the STR variable:

- Name: SUB1
- Value: "to Research"

Create a substring of STR variable:

- Name: SUB2:
- Value: "Com"

```
$ STR="Welcome to Research Computing"
$ echo $STR
Welcome to Research Computing
$ SUB1=${STR:8:11}
to Research
$ SUB2=${STR:20:3}
Com
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Bash allows for simple integer arithmetic:

`(( EXPRESSION ))` or `let VAR=EXPRESSION`
Spaces around `EXPRESSION` do not matter. Dereferencing variables in `EXPRESSION` is optional. There is no overflow checking, except for division by 0.

```
$ x=1
$ y=$((x+2))
$ echo $y
3

$ y=$(($x+2))
$ echo $y
3

$ let y=$x+2
$ echo $y
3
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise:

- Create a variable named X and assign the value 10.
- Create a variable named Y and assign the value 3*X with (( )).
- Create a variable named Z and assign the value as X*Y with let.
- Create a variable named W and assign the value as X+Z.

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise:

- Create a variable named X and assign the value 10.
- Create a variable named Y and assign the value 3*X with (( )).
- Create a variable named Z and assign the value as X*Y with let.
- Create a variable named W and assign the value as X+Z.

Answers:

```
$ X=10
$ Y=$((X*3))
$ echo $Y
30
$ let Z=$X*$Y
$ echo $Z
300
$ W=$(($X+$Z))
$ echo $W
310
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Command substitution:

- Commands between backticks `` are replaced by the output of the command, minus the trailing newline characters
- `variable=$(command)`, saving the output of a command into a variable

```
$ date
Wed Feb 24 14:11:45 EST 2016

$ x=`date`
$ echo $x
Wed Feb 24 14:12:10 EST 2016

$ x=$(date)
$ echo $x
Wed Feb 24 14:12:25 EST 2016
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Escape characters remove the special meaning of a single character that follows. Bash uses a non-quoted backslash \ as the escape character.

Example: using \ to remove the special meaning of $ (dereference the variable `year`):

```
$ year=2016
$ echo $year
2016

$ echo \$year
$year
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Double quotes " " preserve the literal value of each character enclosed with the quotes, except for $, backticks ` `, and backslash \

A " may occur between " ", by preceding it with \

$ and ` ` retain their special meaning within double quotes

```
$ year=2016
$ echo "$year"
2016

$ echo `date`
Wed Feb 24 14:11:45 EST 2016
$ echo "`date`"
Wed Feb 24 14:12:10 EST 2016

$ echo "\\"
\
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Single quotes ' ' preserve the literal value of each character enclosed with the quotes.
A ' may not occur between ' ', even when preceded by \

```
$ year=2016
$ echo $year
$ 2016

$ echo '$year'
$year
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise:

- Test the difference of single and double quotes in your terminal

```
$ STR1="Hello World"
$ LSTR1="MORE   $STR1"
$ LSTR2='MORE   $STR1'
$ echo $LSTR1
MORE Hello World
$ echo "$LSTR1"
MORE   Hello World
$ echo "$LSTR2"
MORE   $STR1
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Shell basics
Shell Types
Variables
String Operations
Arithmetic Operations
Command Substitution
Quoting Characters
Conditionals and Loops

Exercise:

- Test command substitution code in your terminal

```
$ SERVERNAME=$(hostname)
$ echo "Running command on $SERVERNAME...."
$ right_now=$(date +"%x %r %Z")
$ time_stamp="Updated on $right_now by $USER"
$ echo "$time_stamp"
```

- Use backticks in the command substitution code

```
$ SERVERNAME=`hostname`
$ echo "Running command on $SERVERNAME...."
$ right_now=`date +"%x %r %Z"`
$ time_stamp="Updated on $right_now by $USER"
$ echo "$time_stamp"
```

Conditionals and Loops

Acknowledgments

Logging In

Text Manipulation

Regular Expressions

Advanced Text Manipulation

Redirects and Loops

Bash Programming

Conditionals and Loops

test [ ] [[ ]]

if; then; elif; else; fi

For loops (slight return)

break continue

Conditionals and Loops

- test [ ] [[ ]]
- if; then; elif; else; fi
- For loops (slight return)
- break continue

Conditionals and Loops

test [ ] [[ ]]

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Is the expression True or False?

String Comparisons

`[[ -n string ]]`	Is string non-zero length
`[[ -z string ]]`	Is string zero length
`[[ string1 = string2 ]]`	Equal
`[[ string1 != string2 ]]`	Not equal
`[[ string1 > string2 ]]`	Sorts after
`[[ string1 < string2 ]]`	Sorts before

Conditionals and Loops

test [ ] [[ ]]

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Is the expression True or False?

Numeric Comparisons – Note the operator syntax!

`[[ int1 -eq int2 ]]`	Equal
`[[ int1 -ne int2 ]]`	Not equal
`[[ int1 -lt int2 ]]`	Less than
`[[ int1 -gt int2 ]]`	Greater than
`[[ int1 -le int2 ]]`	Less than or equal
`[[ int1 -ge int2 ]]`	Greater than or equal

Conditionals and Loops

test [ ] [[ ]]

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Is the expression True or False?

File and directory conditions

`[[ -d string ]]`	Is string the name of a directory?
`[[ -f string ]]`	Is string the name of a file?
`[[ -r string ]]`	Is string the name of a readable file?
`[[ -w string ]]`	Is string the name of a writable file?
`[[ -x string ]]`	Is string the name of an executable file?
`[[ -s string ]]`	Is string a file with non-zero size?

Conditionals and Loops

if; then; elif; else; fi

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

if *condition*; then *do-this*; fi
if *condition*; then *do-this*; else *do-that*;fi

```
if condition
    then do-this
elif other-condition
    then do-that
else do-other-thing
fi
```

Conditionals and Loops

if; then; elif; else; fi

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Condition can be result of `[[ ]]`

```
#!/bin/bash
if [[ 3 -eq 4 ]]
then
        echo "Not Really"
else
        echo "Math does work"
fi
```

Condition can also be success or failure of program.

```
#!/bin/bash
if grep -q "Laertes" Hamlet.txt
then
        echo "yes it's there"
else
        echo "no Laertes in sight"
fi
```

Conditionals and Loops

For loops (slight return)

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

```
#!/bin/bash

for arg in list
do
Commands ...
    More commands ...
    if [[ $MOOD = "I feel like it" ]]
    then
        Compound commands ...
    fi
        And other stuff ...
done
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Examples:

```
#!/bin/bash
for name in "Bill" "Joe" "Mary"
do
        echo "Hi there, $name"
done
```

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Examples:

List can be a variable:

```
#!/bin/bash

FILES="Hamlet.txt Lear.txt"
SEARCH="Laertes"

for filename in $FILES
do
    if grep -q $SEARCH $filename
    then
        echo "Found \"$SEARCH\" in $filename"
    else
        echo "No \"$SEARCH\" in $filename"
    fi
done
```

Conditionals and Loops

For loops (slight return)

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Examples:

List can be result of command – THIS IS USEFUL!!

```
#!/bin/bash

SEARCH="Laertes"

for filename in $(ls *.txt)
do
    if grep -q "$SEARCH" $filename
    then
        echo "Found \"$SEARCH\" in $filename"
    else
        echo "No \"$SEARCH\" in $filename"
    fi
done
```

Conditionals and Loops

break continue

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

break completely quits a loop.

```
$ for i in {1..25};do if [[ $i -eq 12 ]];then break;fi;echo $i;done
```

With better style:

```
$ for i in {1..25}; do
> if [[ $i -eq 12 ]]
> then break
> fi
> echo $i
> done
```

Notice that semicolon in the second example? Why is it there?

Conditionals and Loops

break continue

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

continue skips to the next item.

```
#!/bin/bash

for i in {1..25}
do
    if [[ $i -eq 12 ]]
    then continue
    fi

    echo $i
done
```

Conditionals and Loops

break continue

Acknowledgments
Logging In
Text Manipulation
Regular Expressions
Advanced Text Manipulation
Redirects and Loops
Bash Programming
Conditionals and Loops
test [ ] [[ ]]
if; then; elif; else; fi
For loops (slight return)
break continue

Exercises: change directory to the Shakespeare directory

1. Print only the names of files whose name (including any extension) is longer than 8 characters.
2. Print only the names of files which are longer than 1000 bytes.

Extra credit Take-Home (there is no credit, sorry)
((**man** and **man -k** are your friends))

- Find the directory in your $PATH variable that contains the largest number of files, and print the directory name and number of files it contains.