

```
File Edit View Terminal Go Help
Search [1]Images [2]Maps [3]Play [4]YouTube [5]News [6]Gmail [7]Drive
[8]More +

[9]Web History | [10]Settings | [11]Sign in

[12][logo_sm_2.gif] naruto Search
[13]Advanced Search
[14]Preferences

Web About 81,400,000 results (0.12 seconds)

[15]Naruto - VIZ.com

Naruto will have to summon all of his Nine Tails power if he hopes to
stand up
against the devastating team of Obito and Madara. Luckily for the
Allied Shinobi ...
www.viz.com/naruto - [16]Similar pages

[17]Naruto - Wikipedia, the free encyclopedia

Naruto (ナルト) is a Japanese manga series written and illustrated by
Masashi
Kishimoto. It tells the story of Naruto Uzumaki, an adolescent ninja
who
constantly ...
en.wikipedia.org/wiki/Naruto - 370k - [18]Cached - [19]Similar pages

[20]Naruto Uzumaki - Narutopedia, the Naruto Encyclopedia Wiki

Naruto Uzumaki (うずまきナルト, Uzumaki Naruto) is a shinobi from
Konohagakure. He became the jinchūriki of Kurama on the day of his
birth, a fate
that caused ...
naruto.wikia.com/wiki/Naruto_Uzumaki - 463k - [21]Cached - [22]Similar
pages

[23]Narutopedia, the Naruto Encyclopedia Wiki - Your Source for
Naruto ...

Narutopedia is a Naruto anime and manga database with info on the
characters,
Shippuden episodes, toys, action figures, Sasuke, Sakura, and Rinata.
https://naruto.wikia.com/ - 516k - [24]Cached - [25]Similar pages

[26]Watch Naruto Online - Free at Hulu

Watch Naruto free online. Stream episodes and clips of Naruto
instantly.
www.hulu.com/naruto - 74k - [27]Cached - [28]Similar pages

[29]Watch Naruto Shippuden Online - Free at Hulu

Watch Naruto Shippuden free online. Stream episodes and clips of Naruto
Shippuden instantly.
```

ATMS 305 WEEK 3: INTRODUCTION TO UNIX/LINUX

Lecture 2: more bash scripting
(yay!)

Adapted from: <http://ryanstutorials.net/bash-scripting-tutorial/bash-script.php>

MORE BASH SCRIPTING TOOLS AND TIPS

Today, we will introduce some more advanced bash scripting techniques that you can use to make scripts more powerful.

INTERACTIVITY

Interactive scripts ask the user for input, and then use that input in the script for something.

In bash, it is easy to generate a prompt in a script.

The read command does the job.

```
read varname
```

will insert the user input into the variable name given

introduction.sh

1. `#!/bin/bash`
2. *`# Ask the user for their name`*
- 3.
4. `echo Hello, who am I talking to?`
- 5.
6. `read varname`
- 7.
8. `echo It\'s nice to meet you $varname`

Terminal

1. `user@bash: ./introduction.sh`
2. `Hello, who am I talking to?`
3. *`Ryan`*
4. `It's nice to meet you Ryan`
5. `user@bash:`

FANCIER READ COMMANDS

login.sh

```
1. #!/bin/bash
2. # Ask the user for login details
3.
4. read -p 'Username: ' uservar
5. read -sp 'Password: ' passvar
6. echo
7. echo Thankyou $uservar we now have your login details
```

s option hides the
input for security

READING MORE THAN ONE VARIABLE AT A TIME


cars.sh

```
1. #!/bin/bash
2. # Demonstrate how read actually works
3.
4. echo What cars do you like?
5.
6. read car1 car2 car3
7.
8. echo Your first car was: $car1
9. echo Your second car was: $car2
10. echo Your third car was: $car3
```

STANDARD INPUT

Standard input is another way to pass information into a script. It uses part of the operating system, and can be used send information in and out of commands and scripts. It is accessible in the filesystem at the location `/dev/stdin`.

```
summary
1. #!/bin/bash
2. # A basic summary of my sales report
3.
4. echo Here is a summary of the sales data:
5. echo =====
6. echo
7.
8. cat /dev/stdin | cut -d' ' -f 2,3 | sort
```



A diagram illustrating the use of a pipe in a shell script. A light blue box labeled "pipe" is connected by a line to the pipe symbol (|) in line 8 of the script, which connects the output of the `echo` command in line 6 to the `cat /dev/stdin` command.

summary

```
1. #!/bin/bash
2. # A basic summary of my sales report
3.
4. echo Here is a summary of the sales data:
5. echo =====
6. echo
7.
8. cat /dev/stdin | cut -d' ' -f 2,3 | sort
```

Terminal

```
1. user@bash: cat salesdata.txt
2. Fred apples 20 February 4
3. Susy oranges 5 February 7
4. Mark watermelons 12 February 10
5. Terry peaches 7 February 15
6. user@bash:
7. user@bash: cat salesdata.txt | ./summary
8. Here is a summary of the sales data:
9. =====
10.
11. apples 20
12. oranges 5
13. peaches 7
14. watermelons 12
```


INPUT INTO SCRIPTS

So we now have 3 methods for getting input from the user:

- Command line arguments (\$1, \$2, \$3, etc.)
- Read input during script execution (read varname1)
- Accept data that has been redirected into the Bash script via STDIN

if_example.sh

```
1. #!/bin/bash
2. # Basic if statement
3.
4. if [ $1 -gt 100 ]
5. then
6.     echo Hey that's a large number.
7.     pwd
8. fi
9.
10. date
```

CONTROL STATEMENTS

Scripts can be set up to perform larger tasks using control statements that allow for decisions to be made based on values of variables or specified numbers of repetitions.

This is done with statements such as

`if/then/else`

`while`

`for`

IF/THEN SYNTAX

if_example.sh

```
1. #!/bin/bash
2. # Basic if statement
3.
4. if [ $1 -gt 100 ]
5. then
6.     echo Hey that's a large number.
7.     pwd
8. fi
9.
10. date
```

OPERATORS USEFUL IN BASH IF STATEMENTS

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The length of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

if_elif.sh

```
1. #!/bin/bash
2. # elif statements
3.
4. if [ $1 -ge 18 ]
5. then
6.     echo You may go to the party.
7. elif [ $2 == 'yes' ]
8. then
9.     echo You may go to the party but be back before midnight.
10. else
11.     echo You may not go to the party.
12. fi
```

MULTIPLE CONDITIONS

AND &&

and.sh

```
1. #!/bin/bash
2. # and example
3.
4. if [ -r $1 ] && [ -s $1 ]
5. then
6.     echo This file is useful.
7. fi
```

OR ||

or.sh

```
1. #!/bin/bash
2. # or example
3.
4. if [ $USER == 'bob' ] || [ $USER == 'andy' ]
5. then
6.     ls -alh
7. else
8.     ls
9. fi
```

MULTIPLE CONDITIONS

AND &&

and.sh

```
1. #!/bin/bash
2. # and example
3.
4. if [ -r $1 ] && [ -s $1 ]
5. then
6.     echo This file is useful.
7. fi
```

OR ||

or.sh

```
1. #!/bin/bash
2. # or example
3.
4. if [ $USER == 'bob' ] || [ $USER == 'andy' ]
5. then
6.     ls -alh
7. else
8.     ls
9. fi
```


WHILE LOOPS — FOR COUNTING WITH ARITHMETIC

while_loop.sh

```
1. #!/bin/bash
2. # Basic while loop
3.
4. counter=1
5. while [ $counter -le 10 ]
6. do
7.     echo $counter
8.     ((counter++))
9. done
10.
11. echo All done
```

while_loop.sh

```
1. #!/bin/bash
2. # Basic while loop
3.
4. counter=1
5. while [ $counter -le 10 ]
6. do
7.     echo $counter
8.     ((counter++))
9. done
10.
11. echo All done
```

Terminal

```
1. user@bash: ./while_loop.sh
2. 1
3. 2
4. 3
5. 4
6. 5
7. 6
8. 7
9. 8
10. 9
11. 10
12. All done
13. user@bash:
```

FOR LOOPS — FOR LOOPING OVER LISTS

```
for_loop.sh
1. #!/bin/bash
2. # Basic for loop
3.
4. names='Stan Kyle Cartman'
5.
6. for name in $names
7. do
8.     echo $name
9. done
10.
11. echo All done
```

lists are separated by spaces

for_loop.sh

```
1. #!/bin/bash
2. # Basic for loop
3.
4. names='Stan Kyle Cartman'
5.
6. for name in $names
7. do
8.     echo $name
9. done
10.
11. echo All done
```

Terminal

```
1. user@bash: ./for_loop.sh
2. Stan
3. Kyle
4. Cartman
5. All done
6. user@bash:
```

OTHER FOR LOOP CONTROLS

RANGES

for_loop_stepping.sh

```
1. #!/bin/bash
2. # Basic range with steps for loop
3.
4. for value in {10..0..2}
5. do
6.     echo $value
7. done
8.
9. echo All done
```

FILE LISTINGS

convert_html_to_php.sh

```
1. #!/bin/bash
2. # Make a php copy of any html files
3.
4. for value in $1/*.html
5. do
6.     cp $value $1/${basename -s .html $value }.php
7. done
```

SUMMARY

Getting user inputs:

- Command line arguments (\$1, \$2, \$3, etc.)
- Read input during script execution (read varname1)
- Accept data that has been redirected into the Bash script via STDIN

Control statements:

if do elseif done – decision based on condition

while do done – loop as long as condition is true

for do done – loop over a specified list or range of integers