Linux Primer

Linux Basics

Objectives

- Introduce Linux
- Files
- Processes
- Features
- Users

Linux Basics

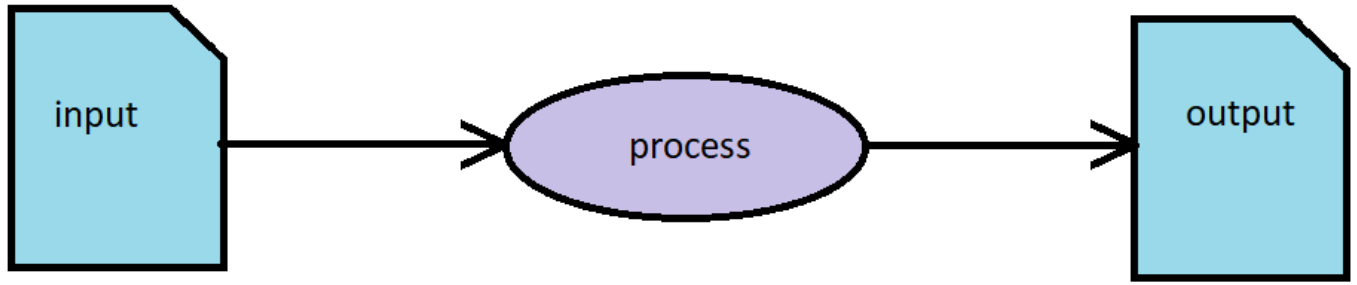
- Major Features
- Key Concepts
- Files & Processes
- Building Pipelines

Major Features

- Simple, powerful, user interface
- Complex commands made from simple
- File system
- Everything is a file
  - Directories too
- Byte stream file format

Key Concepts

- Processes take input and produce output
- Input and output
  - Files
- Programs
  - Processes

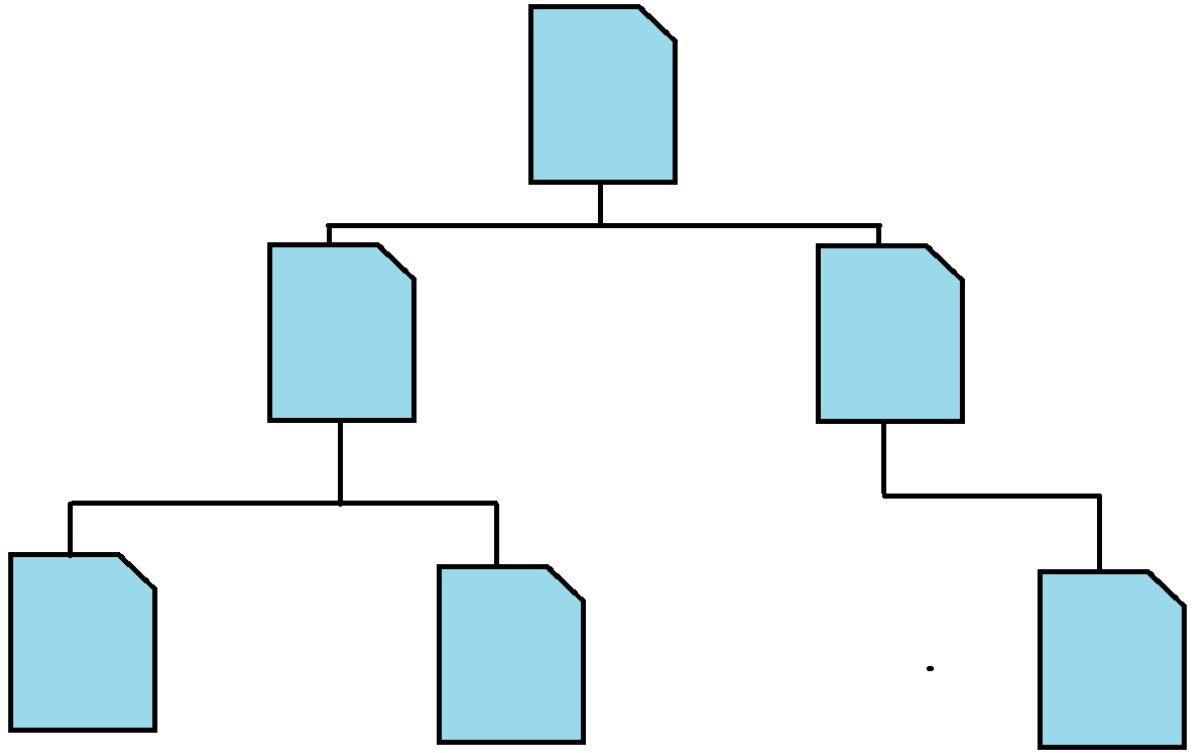


Files and Processes

- Everything is a file
- Some files have execute permissions
  - When run these are a process
- Other files are passive
  - Streams of data

File Organization

- Hierarchy
  - From single root



Process Organization

- Processes have a parent – child organization
- Login to initial shell process
- Subsequent command create new process as child

User

- Owner of files and processes
- Has two IDs
  - User (UID)
  - Group (GID)

Working with Linux

Objectives

- Logging In/Out
- Basic Commands
- File System Commands
- Documentation
- Command Format
- Wildcards
- Redirect

Logging In/Out

- Logging In

  login: fred
  password:

  Last login: Mon Jun 23 13:03:24 on console

  $

- Logging out

  exit
  or
  ^D

Basic Commands

- Getting basic information from the operating system

```
$ pwd
/home/fred

$ ls
data  error inventory.ini scripts

$ date
Mon, Jul 23, 2018   1:56;12 PM

$ who
fred      ttyp4     Jul 23 10:31
augie     ttyp6     Jul 23 09:18
```

Keyboard Commands

- Useful keyboard characters

  ^C command interrupt

  ^Z suspend command

  ^D end of file stream

File System Commands

- Print working directory

  pwd

- Change directory

  cd /
  cd
  cd /etc

- List directory

  ls

- Print content

  cat inventory.ini

Documentation

- Man pages
  - Help from developers of distro
- Specific topic

  man pwd

- Category search

  man –k text

Command Format

- Enter a command with the following format

  command [-options] [files …]

- Reading syntax
  - [] means optional
  - Linux is CASE SENSITIVE
  - Options are a single dash followed by any option character
- Commands are short and simple

Wildcards

- Wildcards are expanded by the shell
  - Can generate file names for commands
    - * any number of characters
    - ? any single character
    - [abc] any of these three characters

```
$ ls
inventory.ini script1.sh script2.sh script3.sh
$ ls s*
script1.sh script2.sh script3.sh
$ ls *.ini
inventory.ini
$ ls *[13].sh
script1.sh script3.sh
```

Redirect

- Output of a command goes to stdout by default
- Redirect to file >
- Redirect to pipe <
- Pipe to next command |

```
$ ls
inventory.ini script1.sh script2.sh script3.sh
$ ls > files
$ cat files
inventory.ini script1.sh script2.sh script3.sh
$ ls | wc -l
```

File System Commands

File System Commands

- Directory Paths
- Navigating the File System
- Looking in the Directory
- Adding and Removing Directories
- File Manipulation
  - Copy
  - Move
  - Delete
- Linking Files

Directory Path

- Root directory named /
- Directory path is absolute or relative
  - Absolute starts with root

    $ cat /home/fred/inventory.ini

  - Relative starts in the current directory

    $ cd /home
    $ cat ./fred/inventory.ini

    or

    $ cat fred/inventory.ini

Navigating the File System

- Two special directory names

  .
  ..

- Change directory command

  $ cd ../fred
  $ cd ..
  $ cd ./report/text

Looking in the Directory

- To list the content of a directory use ls command

  man ls

- Many options

  -F
  -a
  -l
  -d

- Combine as needed

  ls -Fal

- Case sensitive

  -r
  -R

Adding and Removing Directories

- Create directories with

  mkdir

- Remove directories with

  rmdir

Copy Files

- To copy files (and directories) use

  cp

- Can use wildcards

  # Copy all of the css files in report/style to the current directory

  $ cp /report/style/*.css .

Move Files

- To move and to rename

  mv source_file_path destination_file_path

  $ mv report/style/app.css .

  $ mv app.css old.css

Delete Files

- To delete files (and directories) use

  rm

  $ rm –i *.css

Linking Files

- Create another name for the same file
  - Files share the same index number
  - If a link exists the file is available
    - Unless symbolic link

ln [-s] name1 name2

$ ls –li
12103423998560305 -rw-r--r-- 2 augie 197609 0 Jul 25 15:01 file1.txt

$ln file1.txt file2.txt
$ls -li
12103423998560305 -rw-r--r-- 2 augie 197609 0 Jul 25 15:01 file1.txt
12103423998560305 -rw-r--r-- 2 augie 197609 0 Jul 25 15:01 file2.txtx

Working with File Content

Display File Content

- Concatenate content to the standard output

cat

```
$ cat file1.txt
This is the content of
file1.txt which is on
several lines.

$ cat file1.txt file2.txt file3.txt
```

Dealing with Long Files

- Paginate file output
- All systems should have

  cat file1.txt | more

- A newer version with scrolling

  cat file1.txt | less

  - Because “less is more”

Concatenate Part of a File

- The head command displays the first 10 lines of a file
  - Can pass option to display a specific number of lines

$ head longtextfile.txt

$ head -4

Concatenate the End

- The tail command displays the last 10 lines
  - Options to control the number of lines

```
$ tail -3 longtextfile.txt

# starting at  line number 50 to the end
$ tail --lines +50 longtextfile.txt
```

Searching in File

- The grep command can search for text patterns
  - Options to control output type

    man grep
- Can use regular expressions
  - <RE> is a pattern

    . Match any character
    * Match zero or more occurrences of the previous character
    ^ Match line start
    $ Match line end
    [abc] Match any one of a, b, or c
    [0-9] Match any character 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

  grep [-options] <RE> [files …]

Sorting Files

- The sort command organizes output
  - Options for numeric data, reversing the sort and which fields to sort on

```
# sort the directory listing by file size showing the top 3 in size
$ ls –l | sort +4rn | head -3
```

Working with Lines

- Commands like diff and uniq can be used to work with the lines within one or more files

```
$ diff file1.txt file2.txt

$ uniq [-cu] duplicatefiles.txt
```

Cutting Fields

- Use the cut command to only display the selected fields
  - Specify the delimiter and the field list

```
$ ls –l | cut –d” “ –f2,4,9
```

Word Count

- The command wc reports the lines, words and characters in a file
  - Options for just words, just lines or just characters

```
$ ls –l | wc
   5      38       241
$ ls –l | wc –l
5
$ ls –l | wc –c
241
```

Translating Characters

- The tr command translates characters
  - Provide input set and output set
  - Character classes

```
$ echo “Hello World” | tr ‘eo’ ‘30’
H3ll0 W0rld
$ echo “Hello World” | tr [:lower:] [:upper:]
HELLO WORLD
```

Finding Files

- The find command recursively search any part of the file system
  - Search can be based on attributes including
    - -name
    - -user
    - -group
    - -size
  - Can print the name of the file
  - With –exec can execute any command on selected files

```
$ find ./ -name fred.* -exec rm {} \;
```

File Access Permission

Access Control

- UID and GID determine access
- File permissions for User Group or Other
- To see permission use ls –l
  - Read – r
  - Write – w
  - Execute - x

```
$ ls –l
total 10
-rw-r--r-- 2 augie 197609  346 Jul 25 15:48 fred.ini
-rw-r--r-- 2 augie 197609  346 Jul 25 15:48 ginger.ini
```

Access Logic

- The system checks the current process and the file
- If the UID matches, then apply User rights
- Otherwise if the GID matches apply Group rights
- Otherwise apply Other rights

Changing Permission

- Use the chmod command to change permissions
  - Numeric notation
  - Symbolic notation

Numeric Notation

- Treat permissions as a bit pattern
  - 4 = read
  - 2 = write
  - 1 = execute
- For each level state the number to be applied

```
# Set –rw-r--r-x for aFile.txt
chmod 645 aFile.txt
```

Symbolic Notation

- Letters for the levels of permission
  - u g o a
- Operators for the desired action
  - + - =
- Letters for the permission
  - r w x

$ chmod u+rw,g=r,o-w aFile.txt

Default Permissions

- Use the umask command to view and set the default permission bits

```
$ umask
22
$ mkdir newdir
$ ls > newfile
$ ls –ld newdir newfile
drwxr-xr-x    1   augie  197609    0   Jul 26  09:29  newdir/
-rw-r—r--     1   augie  197609   72   July 26 09:30 newfile

r w x    r w x    r w x
4 2 1    4 2 1    4 2 1
           2        2        22 umask
 4+2+0  4+0+0  4+0+0   644 default for files
```

Changing Ownership

- For files use
  - chown
  - chgrp
- For processes
  - login
  - newgrp
  - su

Switch User

- The su command allows one user to temporarily become another user

```
$ id
uid=197609(augie) gid=197609
$ su fred
password:
$ id
uid=197608(fred) gid=197609
$ su
password:
$ id
uid=0(root) gid=1
```

Editors

Objectives

- Linux Editors
  - ed
  - vi

Linux Editors

- Built in Linux edtors
  - ed
    - Interactive editor
    - Buffered
    - Line editing
  - vi
    - Interactive editor
    - Buffered
    - Screen editing
  - sed (discussed later)
    - Non-interactive editor
    - Not buffered
    - Stream editing (inline)

Line Editor

- The line editor is ed
- Available on all distros
  - Original Unix editor
- Can be used in shell scripts to edit files automatically

Starting ed

- Start the editor
- Issue editing command
- Quit and/or save the buffer

```
$ touch sometext
$ ed sometext
0
a
This is the first line
This is the second line
This is the third line
.
W
70
q
```

Commands

- Basic ed commands

i	insert
a	append
c	change
d	delete
w	write
r	read
s/RE/RS/g	substitute RE with RS
t addr	transfer to addr
q	quit
Q	really quit

Using Commands

- Commands have the following structure

[addr[,addr]]<character command> [parameters]

/first/,/second/q d
find the regular expression (first line) for the first address
find the regular expression (second line) for the second address
delete them both

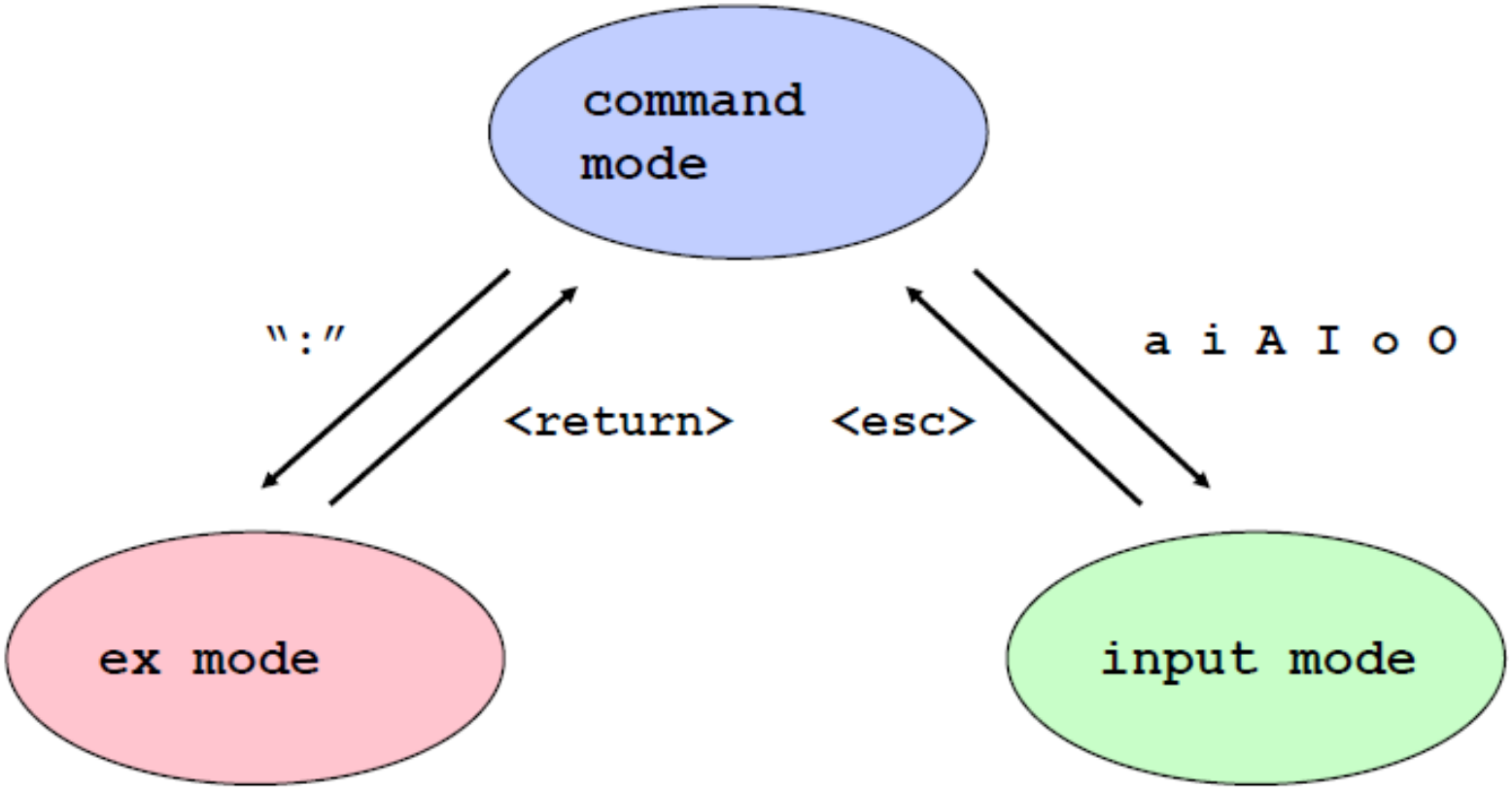
Screen Editor

- The screen editor is vi
  - Most widely used Linux editor
  - An improved version may exist

    vim

  vi [filename]
- Three modes
  - Command mode
  - The ex mode
  - Input mode

Switch Modes



Command Mode

- Like most Linux vi is case sensitive
- Command can be repeated and may take arguments

[n] command [arg]

* repeat the command n times
* apply arg to the command (command specific)

Text Navigation Commands

- Commands allow the text buffer

h	cursor left
l	cursor right
j	cursor down
k	cursor up
w	word forward
b	word backward
^	start of line
$	end of line
G	go to line

Examples:

8h	move left 10 characters
2G	go to line 2
G	go to last line
5w	move forward 5 words

Input Mode

- Command mode switch to Input mode

i	insert before cursor	I	insert at start of line
a	insert after cursor	A	insert at end of line
o	open line below	O	open line above
r	replace character	R	overwrite rest of line
x	delete current character	X	delete character before cursor
d	delete text	D	delete the rest of the line
p	paste buffer before	P	paste buffer after
y	yank text into buffer		

Search

- Commands to search the test frequently use regular expressions

/<RE>	search forward for RE	.	repeat last change
?<RE>	search backward for RE	u	undo last change
n	repeat search	N	reverse search

/th[ae] search forward for tha or the patterns

Switch to ex mode

- Switch from command mode to ex mode with :
- Enter line editing commands

```
:1,10d  #delete lines 1 to 10
:r insert1 #read insert1 in at the current position
```

Customizing vi

- To set preference for every time provide .exrc file
- The vi editor can be customized
  - Use set from the ex mode

```
:set all          # to see the current options
:set autoindent   # indent each line automatically
:set number       # show line numbers
```

Exit vi

- Getting out is an important command
- In command mode

  ZZ

- In ex mode

```
:wq       #write and then quit
:x        # same as wq
:q        # quit – will display error if changes not saved
:q!       # force quit without saving
```

Shell

Linux Shell

- A wrapper around the operating system kernel
  - A shell takes user input
  - Translates input into system instructions
  - Provides resulting output to the user

Standard Streams

- Standard input (STDIN) accepts input
- Standard output (STDOUT) sends output back to the user
- Error messages are sent view (STDERR)
  - Typically STDIN = keyboard, STDOUT – terminal and STDERR = termal
  - There can be redirected

Redirection

- A common redirect is the STDOUT with >
  - >> to append
- The input (STDIN) can also be redirected with <
- Redirecting STDERR using 2>
- Redirecting STDERR to the same as STDOUT

```
$ ls –l > filelists
$ ls file1.txt file2.txt missingfile.txt > listing2 > 2>errors
$ ls file1.txt file2.txt missingfile.txt > listing2 2>&1
```

Accepting Multiline Input

- Redirect STDIN with << END
  - END on a line by itself will stop the redirect

```
$ mail someone@somedomand.com << END
Now everything that I type
will be captured and passed in
to the current command until
I want to END
by typing the work END
on a line of its own.
END
```

Piping between commands

- The STDOUT of one command becomes the STDIN of the next command
  - STDERR is send to the terminal unless specifically redirected

```
$ ls –l | cut –d “ “ –f3-5 | grep [ae]u

# the output of the directory listing is taken by the cut command
# cut uses space delimiters and outputs fields 3 through 5 to
# the grep command with looks for lines that contain the
# ‘au’ or ‘eu’ patterns
```

Shell Command Processing

- Shell reads the line
- Parses the input
- Processes directives (like redirect)
- Runs the actual command
- Each command spawns a new process
  - The pid is the process ID
  - The ppid is the parent process ID
  - Child inherits parent attributes
  - Child returns 0 if successful

Variables

- String variables can be created
  - Can be interpreted as a number
- System or user set
- Can be local or global
- By convention use all uppercase names

Important Shell Variables

- To display current variables use set for local variables and env for environment variables

  $ set
  $ env

- PATH
  - Path search for commands
- EDITOR
  - The path to the default editor
- SHELL
  - The shell you are currently using
- HOME
  - Path to your home directory

Creating Variables

- Create a variable by naming it and assigning a value
- Local variables are not seen in a child process
- Use the export command to change variables to global

```
$ FILENAME = sometext
$ export FILENAME
```

Alias

- Using alias you can define one string to mean something else

```
$ alias  # to list current values
$ alias dir=“ls –l”
$ dir
$ unalias dir
```

Shell Programming

Shell Programming

- Shell programs are called scripts
  - Text files that contain collections of commands
- Bash shell is very popular in Linux

```
$ cat firstscript.sh
#!/bin/bash
ls
pwd

$ ./firstscript.sh
```

Executing a Script

- Must be able to find the script
  - PATH variable
  - Or in the current directory

  $ ./firstscript
- The script must have execute permission for the user, group or others

  $chmod u+x firstscript

Command Line Arguments

- When a script is run it can be passed arguments
- The script can access ten arguments with $0 through $9
- $# returns the number of arguments
- $* or $@ shows all arguments
  - $* treats everything as a single argument
  - $@ keeps separate arguments
- $0 is always the command
- The shift command can be used to move the everything to the left, getting what would be $10 and putting it in $9
  - $1 is replaced with $2, etc

Getting User Input

- The read command reads from STDIN
  - Loading each of the variables provided
  - Loading the last variable with any remaining input

```
#!/bin/bash
echo “What’s your name? “
read first last other
echo hi $first $last
echo “what’s $other”
```

```
$ ./firstscript
What’s your name? Augie Schau Augiesson
```

Conditional Commands

- Using if statement
  - If condition is true executes
  - Ends with fi
  - Optional else

```
if mkdir test
then
        cd test
else
        echo “No directory named test is available”
fi
```

- For multiple tests optional elif with its own command

Testing File Conditions

test –option filename

-d True if file is a directory

-f True if file exists and is a regular file

-r True if file is readable by you

-s True if file exists and is not empty

-w True if the file is writable by you

-x True if the file is executable by you

Testing Number or String

test n1 –eq n2 other number comparisons (-ne, -ge, -le, -lt, -gt)

test str1 = str2 other string comparisons (!=, >, <)

[ ] is an alias for test

[n1 –eq n2] notice the spaces

Multiple Test Case

- The case statement checks a value and provides multiple blocks for each of the cases
  - Patterns can have wildcards
  - Ends with esac

```
case MYVARIABLE in
        pattern1)       commands
                        ;;
        pattern2)       commands
                        ;;
        *)              commands
                        ;;
esac
```

The for Loop

- Iterates through each item in a list of values

```
for f in *
do
      echo “File: $f”
done


# list out the files in the current directory
```

Command Generated List

```
for f in `cat filelist`
do
        echo “This is the content”
        echo $f
        echo “…”
done
```

The while Loop

- Use the while loop when there is some flag to control the loop

```
# This example processes all the arguments passed to the script
while [ “$1” != “end” ]
do
        echo $1
        shift
done
```

Evaluate Expressions

- The expr command evaluates expressions
  - Be careful with spaces between arguments

```
echo –n “Enter two numbers: “
read one two
echo The sum is `expr $one + $two`
```

Jumps

- The break command ends a loop before it is logically finished
- The continue command stops the current iteration and goes to the next item