4. `` `if` ``: Conditionally executes a block of code based on a condition.
5. `` `for` ``: Executes a block of code for each item in a list.
6. `` `while` ``: Executes a block of code repeatedly as long as a condition is true.
7. `` `case` ``: Evaluates a variable against a list of patterns and executes code based on matching pattern.

Properties:

1. `` `$0` ``: The name of the script or the currently executing command.
2. `` `$#` ``: The number of command line arguments passed to the script or function.
3. `` `$@` ``: All the command line arguments passed to the script or function as an array.
4. `` `$?` ``: The exit status of the last executed command.
5. `` `$$` ``: The process ID (PID) of the current script.

This is just a brief overview, and there are many more features and functionalities available in the Bash scripting language. 😄

Hi

give me an example of each point you sent while maintaining the explanation

TheB.AI Free

Certainly! Here are examples for each point mentioned:

Syntax:

1. Comments:

bash

```bash
# This is a comment
```

1. Variables:

bash

```
name="John Doe"
echo $name
```

1. Command substitution:

bash

```
current_date=$(date +%Y-%m-%d)
echo "Today's date is: $current_date"
```

1. Conditional statements:

bash

```
if [ "$x" -gt 10 ]; then
    echo "x is greater than 10"
else
    echo "x is less than or equal to 10"
fi
```

1. Loops:

bash

```
for number in 1 2 3 4 5
do
    echo $number
done
```

1. Functions:

bash

```
# Function defined using the function keyword
function greet() {
    echo "Hello, $1!"
}

# Function defined without the function keyword
bye() {
    echo "Goodbye, $1!"
}

greet "Alice"
bye "Bob"
```

1. String manipulation:

bash

```bash
name="John Doe"
echo ${name:0:4}  # Output: "John"
```

1. Arithmetic operations:

bash

```bash
x=5
y=$((x + 10))
echo $y  # Output: 15
```

1. Input/Output redirection:

bash

```bash
# Reading input from a file
while read line
do
    echo "$line"
done < input.txt

# Appending output to a file
echo "New content" >> output.txt
```

1. Pipelines:

bash

```bash
# Filtering output using a pipeline
ls -l | grep ".txt"
```

Functions:

1. **`echo`**:

bash

```bash
echo "Hello, World!"
```

1. **`read`**:

bash

```bash
echo "What is your name?"
read name
echo "Hello, $name!"
```

1. **`printf`**:

bash

```bash
printf "Today is %s\n" $(date +%A)
```

1. **`if`**:

bash

```bash
x=20
if [ "$x" -gt 10 ]; then
    echo "x is greater than 10"
else
    echo "x is less than or equal to 10"
fi
```

1. **`for`**:

bash

```bash
for animal in cat dog lion
do
    echo "I love $animal"
```

```
done
```

1. **`while`**:

```bash
counter=1
while [ $counter -le 5 ]
do
    echo "Count: $counter"
    ((counter++))
done
```

1. **`case`**:

```bash
fruit="apple"
case $fruit in
    "apple")
        echo "It's a delicious apple!"
        ;;
    "banana")
        echo "It's a ripe banana!"
        ;;
    *)
        echo "Unknown fruit!"
        ;;
esac
```

Properties:

1. **`$0`**:

```bash
echo "Script name: $0"
```

1. **`$#`**:

bash

```bash
echo "Number of arguments: $#"
```

1. **`$@`**:

bash

```bash
echo "All arguments: $@"
```

1. **`$?`**:

bash

```bash
ls non_existent_file
echo "Exit status: $?"
```

1. **`$$`**:

bash

```bash
echo "Process ID: $$"
```

I hope these examples help! 😊