Variables and other tricks in Bash

ComS 252 — Iowa State University

Andrew Miner

Warning

- The syntax for this lecture is specific to `bash`
- Other shells may support some of these things
  - Sometimes using the same syntax
  - Sometimes using a different syntax

Defining

To define a variable in bash:

```
VARNAME=value
```

Note:

- Variables have no type
  - Or, if you prefer, they all have type string
- Variable names must start with a letter or underscore
- Variable names may contain digits
- Variable names are case sensitive

Using

To use a variable in `bash`:

```
$VARNAME
```

Note:

- If the variable has not been defined, you get empty string

Using

To use a variable in `bash`:

```
$VARNAME
```

Note:

- If the variable has not been defined, you get empty string

You can also use *substrings* of variables in `bash`:

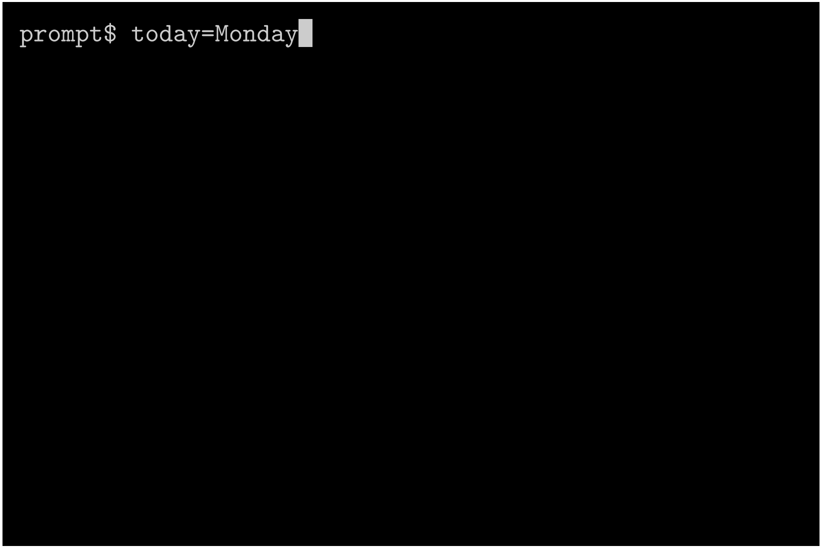
${VARNAME:n}
: *Use variable VARNAME but discard the first n characters*

${VARNAME:n:m}
: *Use variable VARNAME but discard the first n characters, and use only the first m characters after that*

Simple example for variables

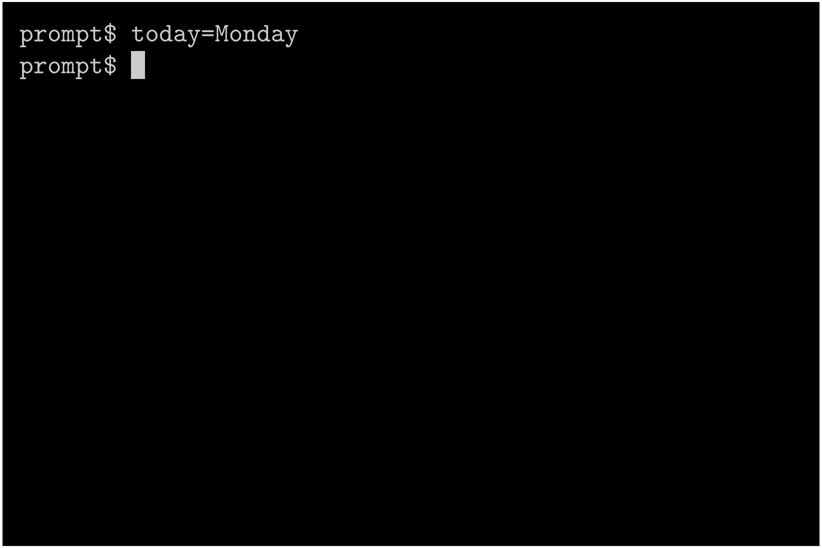
```
prompt$ 
```

Simple example for variables



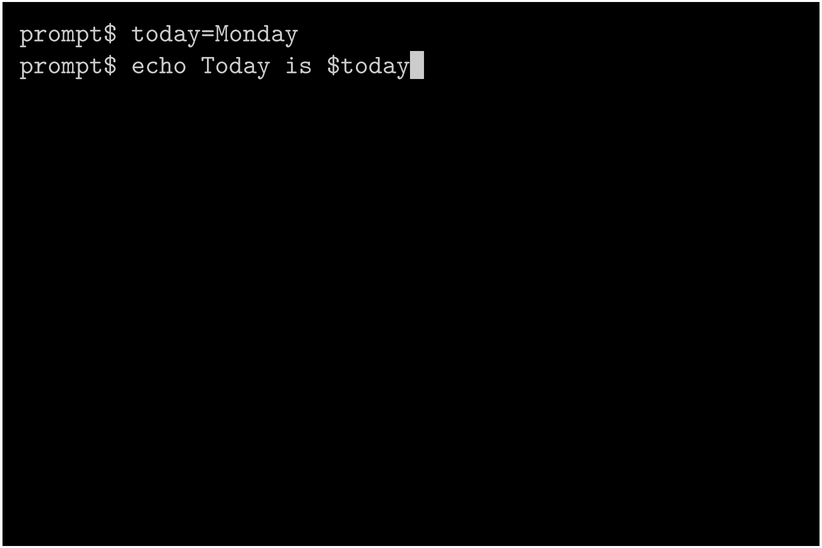
```
prompt$ today=Monday
```

Simple example for variables



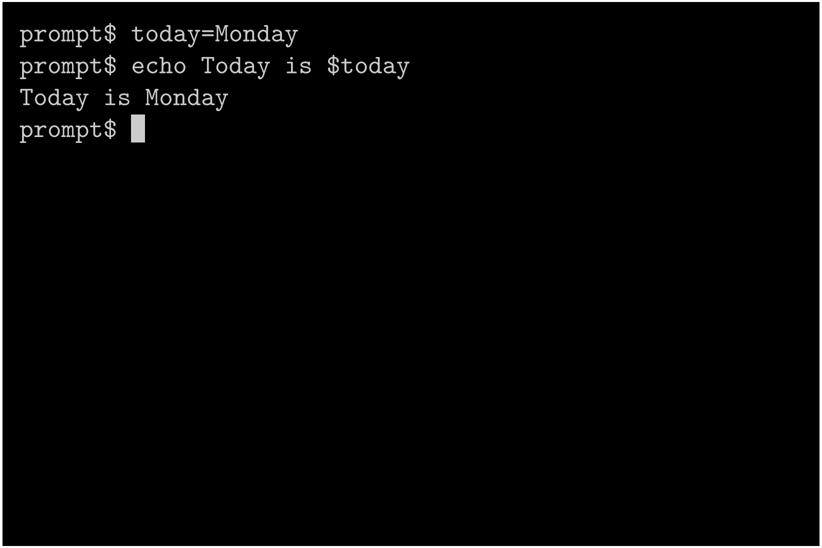
```
prompt$ today=Monday
prompt$ 
```

Simple example for variables



```
prompt$ today=Monday
prompt$ echo Today is $today
```

Simple example for variables



```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ 
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ █
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ █
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ █
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ today=Monday
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ today=Monday
prompt$ █
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ today=Monday
prompt$ !e
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ today=Monday
prompt$ !e
echo Tomorrow is $tomorrow
Tomorrow is Tuesday
prompt$ █
```

Simple example for variables

```
prompt$ today=Monday
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ today=Monday
prompt$ !e
echo Tomorrow is $tomorrow
Tomorrow is Tuesday
prompt$ echo Tomorrow shorthand is ${tomorrow:0:3}█
```

Simple example for variables

```
prompt$ echo Today is $today
Today is Monday
prompt$ today=Tuesday
prompt$ !e
echo Today is $today
Today is Tuesday
prompt$ echo Tomorrow is $tomorrow
Tomorrow is
prompt$ tomorrow=$today
prompt$ today=Monday
prompt$ !e
echo Tomorrow is $tomorrow
Tomorrow is Tuesday
prompt$ echo Tomorrow shorthand is ${tomorrow:0:3}
Tomorrow shorthand is Tue
prompt$ █
```

Reading values from standard input

`read`: read values into variables

- Usage: `read var1 var2 ...var`*n*
  - Reads a line from standard input
  - The first word goes into `var1`
  - The second word (if any) goes into `var2`

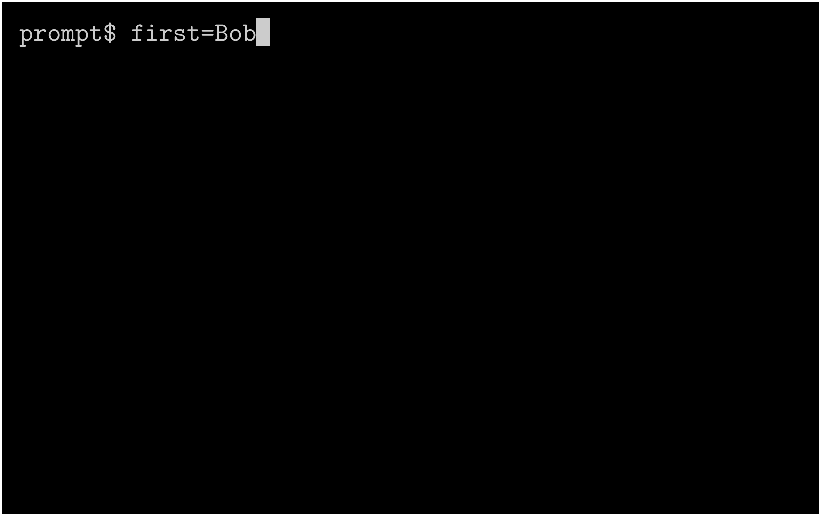
    ⋮
  - Any remaining words go into `var`*n*
- Variables `var1`, ..., `var`*n* can be existing or not

`-p` : Specify a prompt

read examples

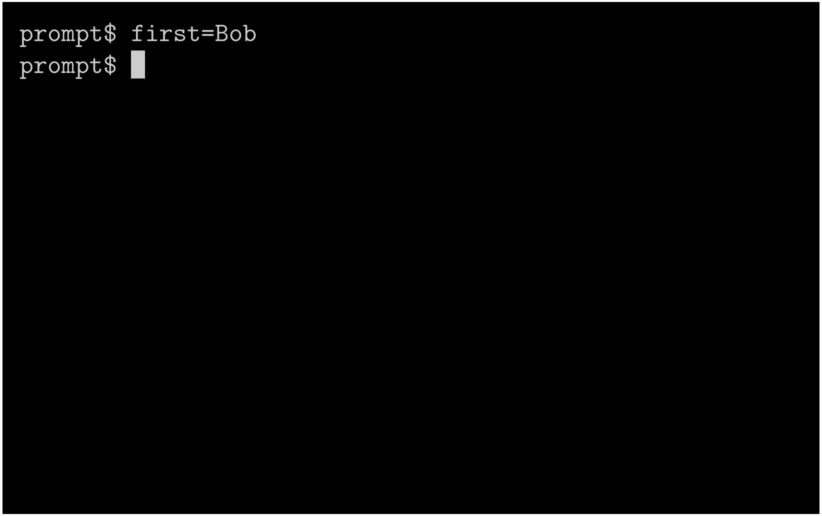
```
prompt$ 
```

read examples



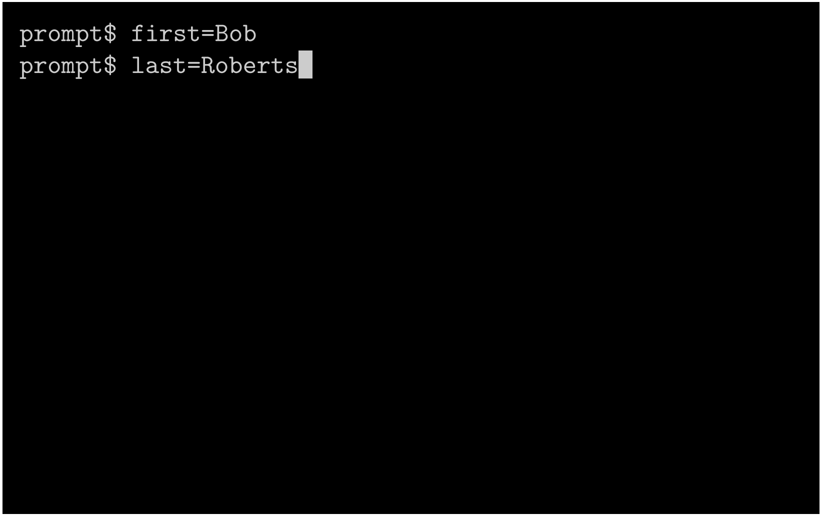
```
prompt$ first=Bob
```

read examples



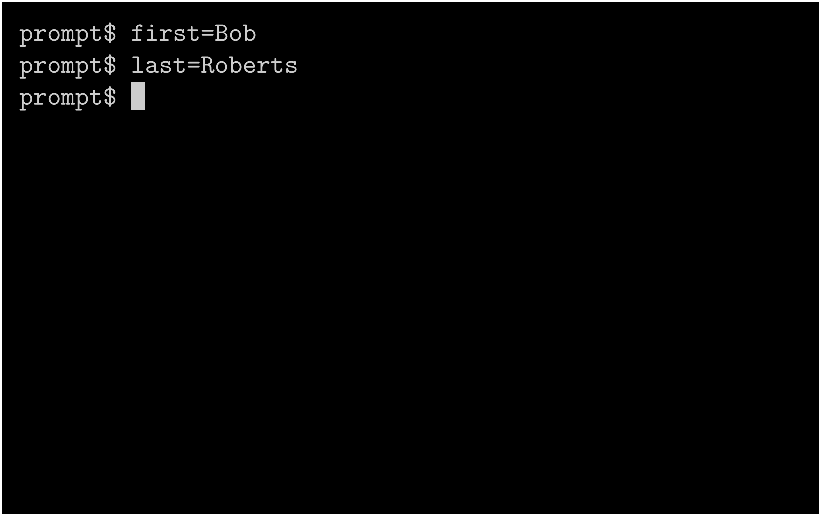
```
prompt$ first=Bob
prompt$ █
```

read examples



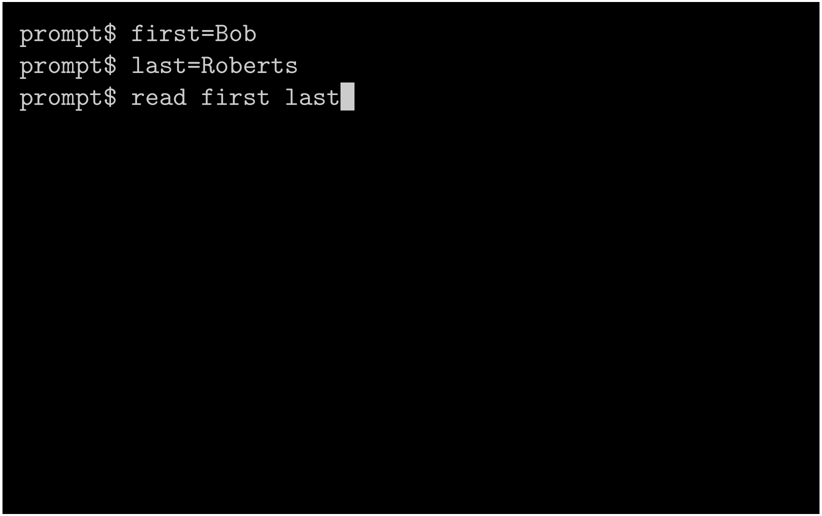
```
prompt$ first=Bob
prompt$ last=Roberts
```

read examples



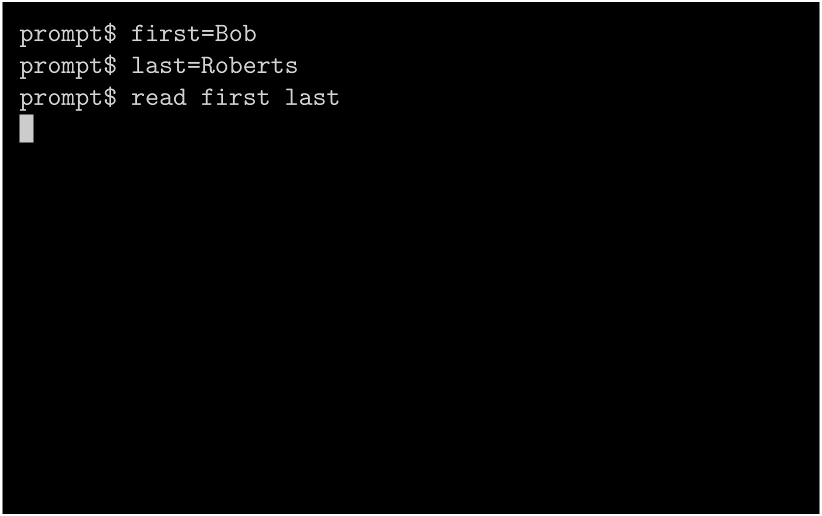
```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ 
```

read examples



```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
```

read examples



```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
█
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
```

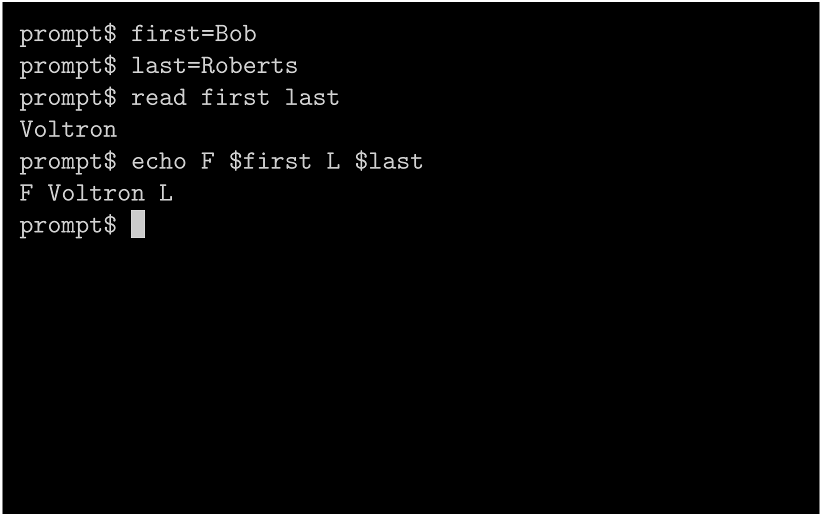
read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ █
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
```

read examples



```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ 
```

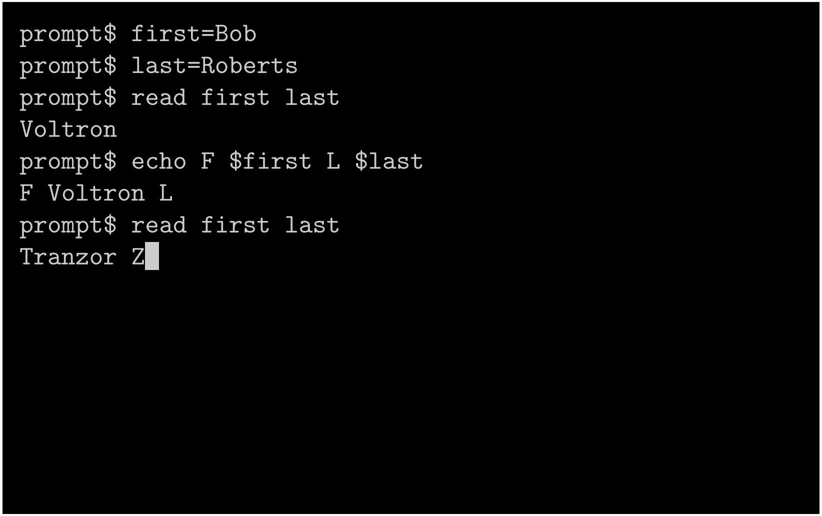
read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
█
```

read examples



```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
```

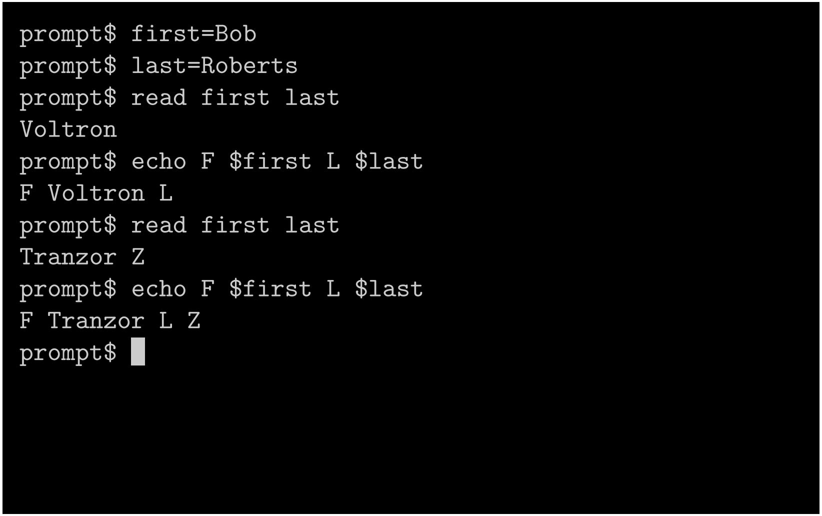
read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ █
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
```

read examples



```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ 
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ read first last
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ read first last
█
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ read first last
The artist formerly known as Prince
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ read first last
The artist formerly known as Prince
prompt$ █
```

read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ read first last
The artist formerly known as Prince
prompt$ echo F $first L $last
```

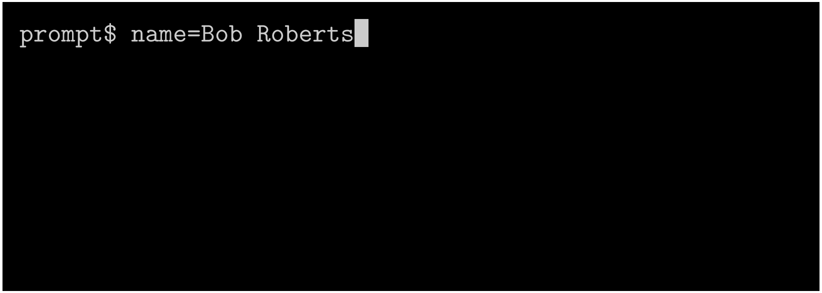
read examples

```
prompt$ first=Bob
prompt$ last=Roberts
prompt$ read first last
Voltron
prompt$ echo F $first L $last
F Voltron L
prompt$ read first last
Tranzor Z
prompt$ echo F $first L $last
F Tranzor L Z
prompt$ read first last
The artist formerly known as Prince
prompt$ echo F $first L $last
F The L artist formerly known as Prince
prompt$ █
```

Motivating variable examples

```
prompt$ █
```

Motivating variable examples



```
prompt$ name=Bob Roberts
```

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
```

Questions:

1. Can a variable be set to a string with spaces?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
```

Questions:

1. Can a variable be set to a string with spaces?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ 
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type “`dir`” instead of “`$dir`”?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type “`dir`” instead of “`$dir`”?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?

Motivating variable examples

```
prompt$ name=Bob Roberts
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ echo $dir
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type “`dir`” instead of “`$dir`”?

Motivating variable examples

```
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ echo $dir

prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?

Motivating variable examples

```
-bash: Roberts: command not found
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ echo $dir

prompt$ exit
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?

Motivating variable examples

```
prompt$ dir=ls
prompt$ $dir
bar.txt    foo.txt    hello.c    Readme
prompt$ bash
prompt$ echo $dir

prompt$ exit
prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?

Motivating variable examples

```
prompt$ dir=ls
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ echo $dir

prompt$ exit
prompt$ i=3
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?

Motivating variable examples

```
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ echo $dir

prompt$ exit
prompt$ i=3
prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?

Motivating variable examples

```
prompt$ $dir
bar.txt   foo.txt   hello.c   Readme
prompt$ bash
prompt$ echo $dir

prompt$ exit
prompt$ i=3
prompt$ echo $i+1
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?

Motivating variable examples

```
prompt$ bash
prompt$ echo $dir

prompt$ exit
prompt$ i=3
prompt$ echo $i+1
3+1
prompt$ █
```

Questions:

1. Can a variable be set to a string with spaces?
2. Can I do something so I can type "`dir`" instead of "`$dir`"?
3. Can I make variables visible to child processes?
4. Can I make variables visible to all shells?
5. Can I do arithmetic in `bash`?

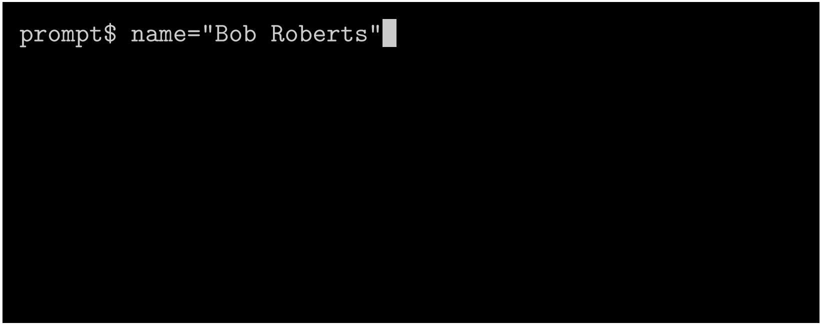
Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ █
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution



```
prompt$ name="Bob Roberts"
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ 
```

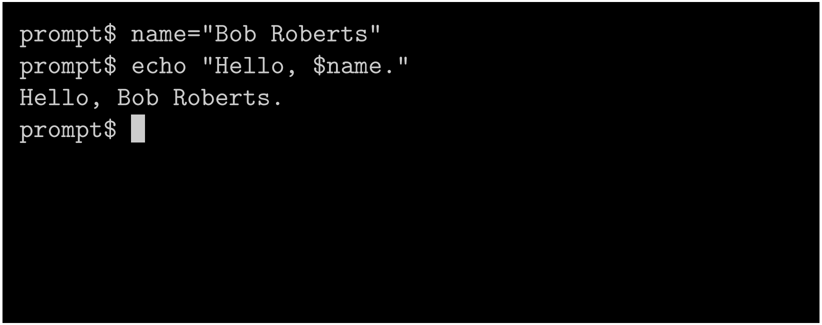
Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution



```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
Hello, Bob Roberts.
prompt$ 
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
Hello, Bob Roberts.
prompt$ ls
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
Hello, Bob Roberts.
prompt$ ls
bar.txt   foo.txt   hello.c   Readme
prompt$ █
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
Hello, Bob Roberts.
prompt$ ls
bar.txt   foo.txt   hello.c   Readme
prompt$ echo "!!"
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
Hello, Bob Roberts.
prompt$ ls
bar.txt   foo.txt   hello.c   Readme
prompt$ echo "!!"
echo "ls"
ls
prompt$ █
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
prompt$ name="Bob Roberts"
prompt$ echo "Hello, $name."
Hello, Bob Roberts.
prompt$ ls
bar.txt   foo.txt   hello.c   Readme
prompt$ echo "!!"
echo "ls"
ls
prompt$ echo 2 * 3
```

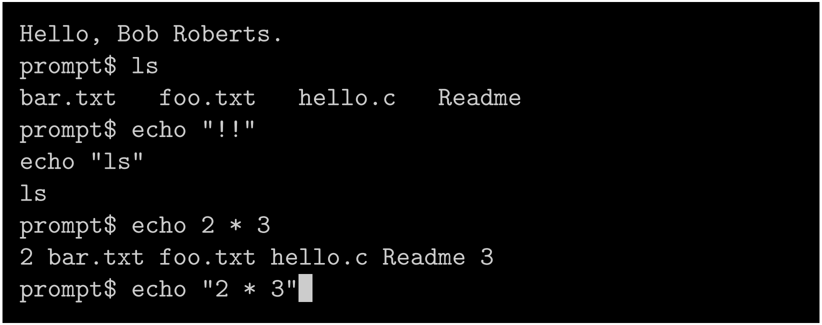
Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
Hello, Bob Roberts.
prompt$ ls
bar.txt   foo.txt   hello.c   Readme
prompt$ echo "!!"
echo "ls"
ls
prompt$ echo 2 * 3
2 bar.txt foo.txt hello.c Readme 3
prompt$ █
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution



```
Hello, Bob Roberts.
prompt$ ls
bar.txt   foo.txt   hello.c   Readme
prompt$ echo "!!"
echo "ls"
ls
prompt$ echo 2 * 3
2 bar.txt foo.txt hello.c Readme 3
prompt$ echo "2 * 3"
```

Double quotes

- Can group things into a single string
- Will prevent *some* shell expansions
- Still allows variable substitution

```
bar.txt   foo.txt   hello.c   Readme
prompt$ echo "!!"
echo "ls"
ls
prompt$ echo 2 * 3
2 bar.txt foo.txt hello.c Readme 3
prompt$ echo "2 * 3"
2 * 3
prompt$ 
```

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ █
```

[1] If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
```

[1] If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ █
```

[1]If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ echo $name
```

[1]If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ echo $name
Bob Roberts
prompt$ █
```

[1] If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ echo $name
Bob Roberts
prompt$ echo 'Hello, $name.'
```

[1] If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ echo $name
Bob Roberts
prompt$ echo 'Hello, $name.'
Hello, $name.
prompt$ █
```

[1]If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ echo $name
Bob Roberts
prompt$ echo 'Hello, $name.'
Hello, $name.
prompt$ echo '!!'
```

[1]If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name='Bob Roberts'
prompt$ echo $name
Bob Roberts
prompt$ echo 'Hello, $name.'
Hello, $name.
prompt$ echo '!!'
!!
prompt$ █
```

[1]If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ name=’Bob Roberts’
prompt$ echo $name
Bob Roberts
prompt$ echo ’Hello, $name.’
Hello, $name.
prompt$ echo ’!!’
!!
prompt$ echo ’2 * 3’
```

[1] If you find one that is not prevented, let me know!

Single quotes

- Can group things into a single string
- Will prevent *all* shell expansions[1]
- Does not substitute variables

```
prompt$ echo $name
Bob Roberts
prompt$ echo ’Hello, $name.’
Hello, $name.
prompt$ echo ’!!’
!!
prompt$ echo ’2 * 3’
2 * 3
prompt$ █
```

[1] If you find one that is not prevented, let me know!

“Backward” quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
`command`
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$
```

“Backward” quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
‘command‘
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=‘date | head -c 10‘
```

“Backward” quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
‘command‘
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=‘date | head -c 10‘
prompt$
```

"Backward" quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
`command`
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=`date | head -c 10`
prompt$ echo $today
```

“Backward” quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
‘command‘
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=‘date | head -c 10‘
prompt$ echo $today
Fri Sep 23
prompt$
```

“Backward” quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
‘command‘
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=‘date | head -c 10‘
prompt$ echo $today
Fri Sep 23
prompt$ echo "The current time is ‘date | tail -c +11 | head -c 9‘"
```

“Backward” quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
‘command‘
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=‘date | head -c 10‘
prompt$ echo $today
Fri Sep 23
prompt$ echo "The current time is ‘date | tail -c +11 | head -c 9‘"
The current time is  12:03:27
prompt$ █
```

Alternate syntax for “backward” quotes

```
$(command)
```

does the same thing as

```
`command`
```

For example:

```
prompt$ █
```

Alternate syntax for “backward” quotes

```
$(command)
```

does the same thing as

```
`command`
```

For example:

```
prompt$ firstfile=$(ls | head -n 1)
```

Alternate syntax for “backward” quotes

```
$(command)
```

does the same thing as

```
`command`
```

For example:

```
prompt$ firstfile=$(ls | head -n 1)
prompt$ █
```

Alternate syntax for “backward” quotes

```
$(command)
```

does the same thing as

```
`command`
```

For example:

```
prompt$ firstfile=$(ls | head -n 1)
prompt$ echo "The first file is $firstfile"
```

Alternate syntax for “backward” quotes

```
$(command)
```

does the same thing as

```
‘command‘
```

For example:

```
prompt$ firstfile=$(ls | head -n 1)
prompt$ echo "The first file is $firstfile"
The first file is bar.txt
prompt$ 
```

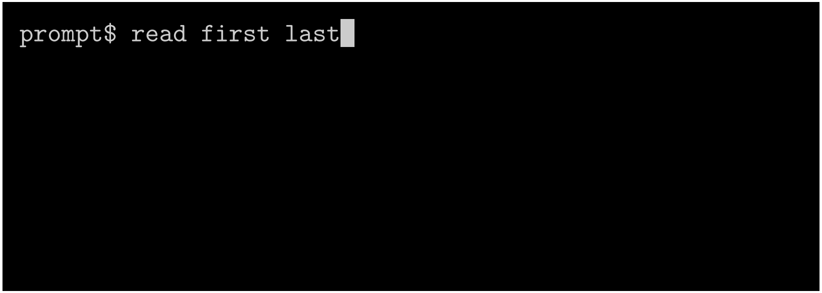
Quotes with read

- Will quotes collect a string together, for read?

```
prompt$ █
```

Quotes with `read`

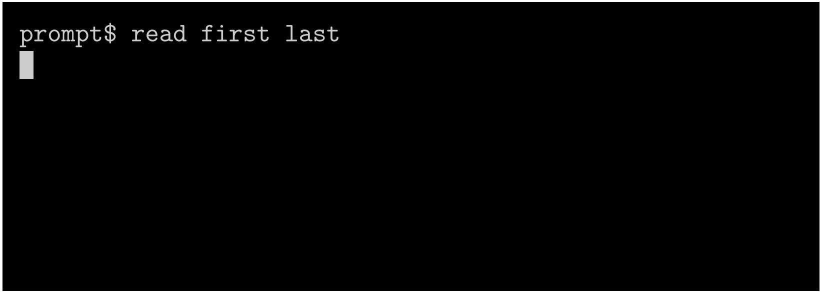
- Will quotes collect a string together, for `read`?



```
prompt$ read first last
```

Quotes with `read`

- Will quotes collect a string together, for `read`?



```
prompt$ read first last
█
```

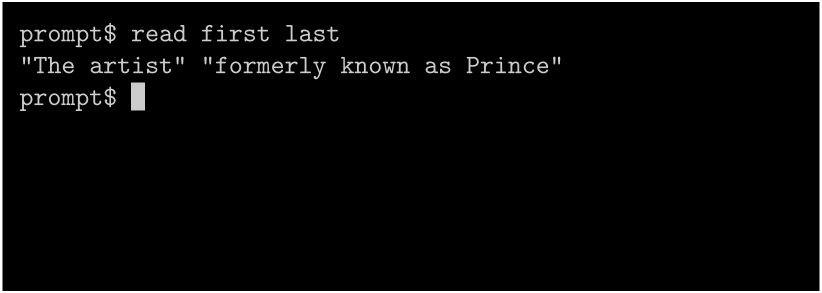
Quotes with read

- Will quotes collect a string together, for read?

```
prompt$ read first last
"The artist" "formerly known as Prince"
```

Quotes with `read`

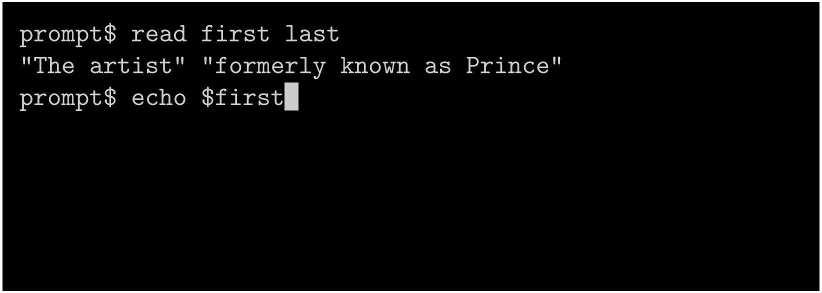
- Will quotes collect a string together, for `read`?



```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ 
```

Quotes with `read`

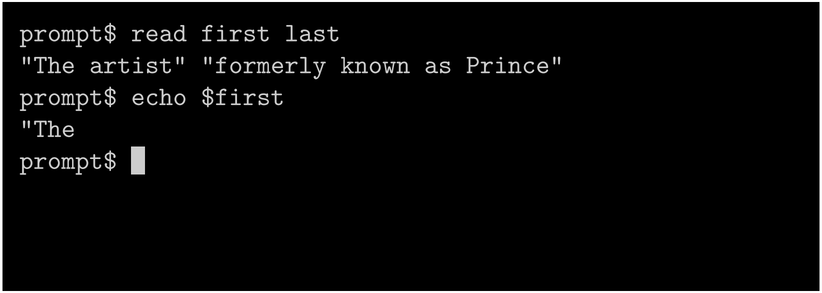
- Will quotes collect a string together, for `read`?



```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
```

Quotes with `read`

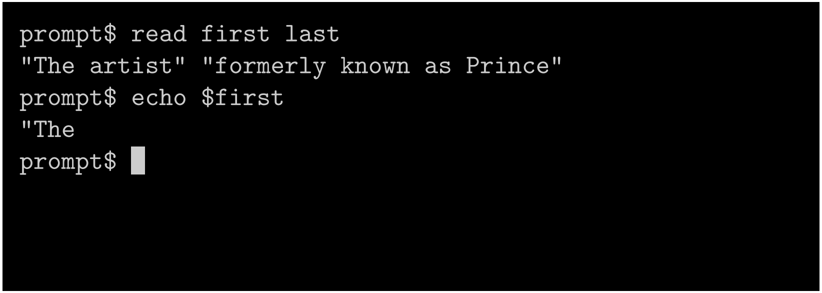
- Will quotes collect a string together, for `read`?
  - No



```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?



```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ 
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ █
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
```

Quotes with read

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
```

Quotes with read

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
prompt$ 
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
prompt$ read first last
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
prompt$ echo $first
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
prompt$ echo $first
The artist
prompt$ █
```

Quotes with read

- Will quotes collect a string together, for read?
  - No
- How can we collect a few words together, for read?
  - Escape the spaces

```
"The artist" "formerly known as Prince"
prompt$ echo $first
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
prompt$ echo $first
The artist
prompt$ echo $middle
```

Quotes with `read`

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
prompt$ echo $first
The artist
prompt$ echo $middle
formerly known as
prompt$ █
```

Quotes with read

- Will quotes collect a string together, for read?
  - No
- How can we collect a few words together, for read?
  - Escape the spaces

```
"The
prompt$ read first middle last
The\ artist formerly\ known\ as Prince
prompt$ echo $first
The artist
prompt$ echo $middle
formerly known as
prompt$ echo $last
```

Quotes with read

- Will quotes collect a string together, for `read`?
  - No
- How can we collect a few words together, for `read`?
  - Escape the spaces

```
The\ artist formerly\ known\ as Prince
prompt$ echo $first
The artist
prompt$ echo $middle
formerly known as
prompt$ echo $last
Prince
prompt$ █
```

What is an alias?

- An *alias* allows us to specify our own (simple) shell builtins
- Use the same naming rule as shell variables
- Are defined similar to shell variables
- May have the same name as an existing "command"
- Are used by typing the name as a "command"
  - Without a leading $
  - Are substituted *only* when treated as a "command"

`alias`: set and display aliases

- `alias`
  - Display all known aliases
- `alias NAME`
  - Display the alias defined for `NAME`
  - Prints an error, if none
- `alias NAME=command`
  - Sets the alias for `NAME`
  - Use quotes to collect a string, as usual
  - May contain other aliases...
  - ...but must eventually resolve to a "real command"

`unalias`: remove an alias

- Usage: `unalias NAME1 NAME2 ...`
- Prints an error, if some `NAME` is not an existing alias

Alias examples

```
prompt$ █
```

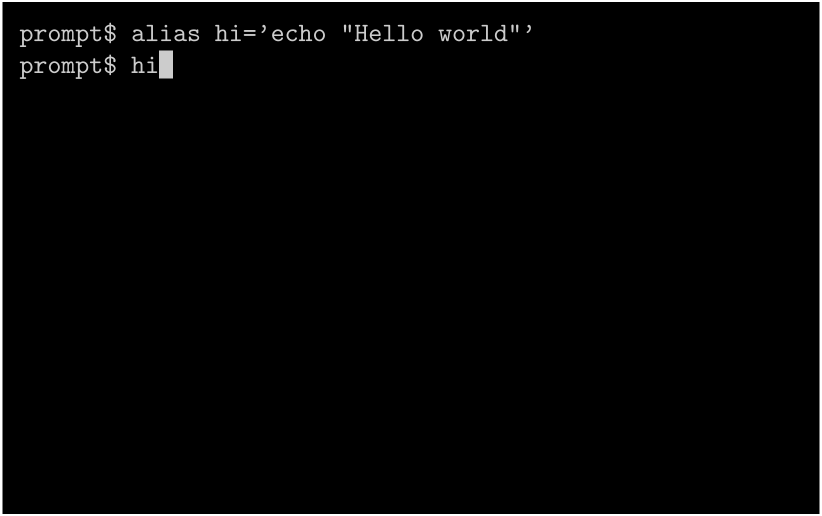
Alias examples

```
prompt$ alias hi='echo "Hello world"'
```

Alias examples

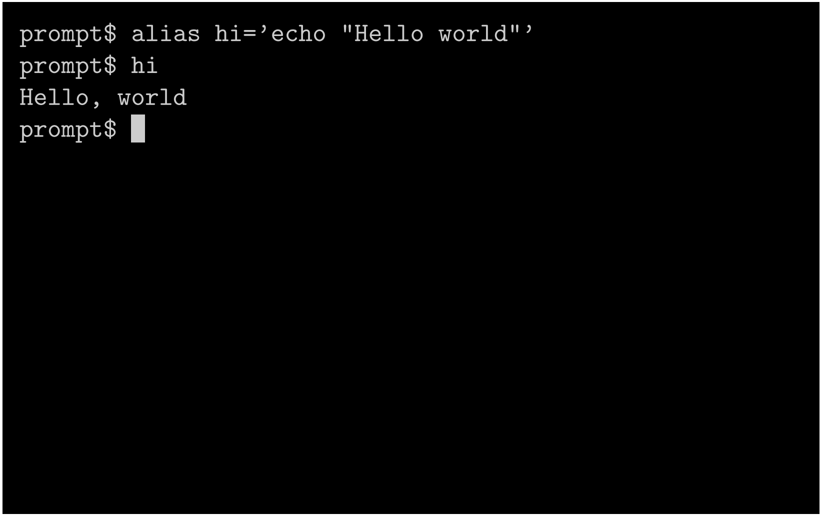
```
prompt$ alias hi='echo "Hello world"'
prompt$ 
```

Alias examples



```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
```

Alias examples



```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ 
```

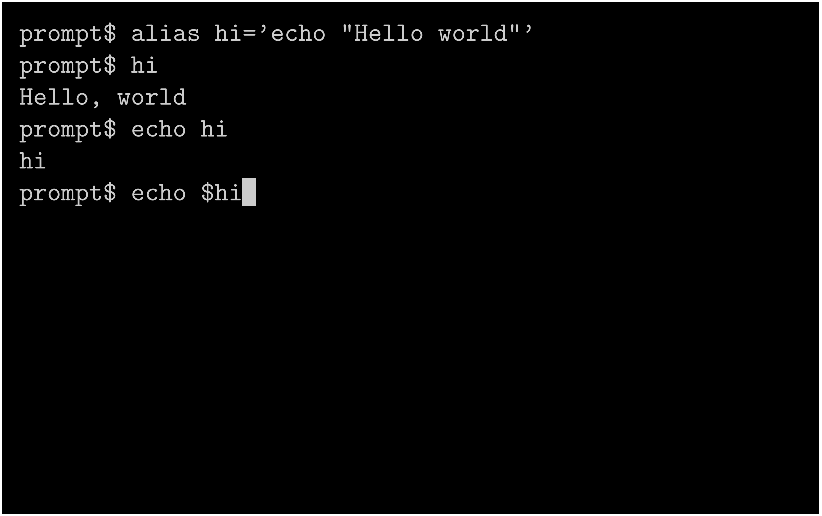
Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
```

Alias examples

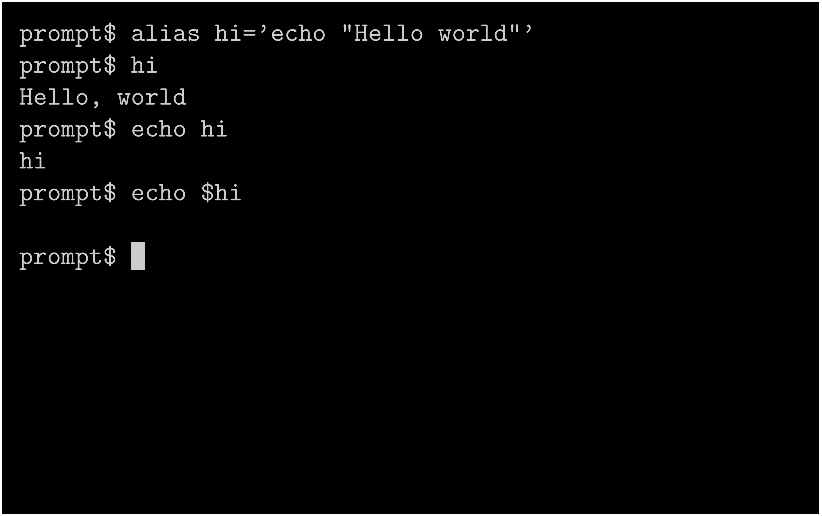
```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ █
```

Alias examples



```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi
```

Alias examples



```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ 
```

Alias examples

```
prompt$ alias hi=’echo "Hello world"’
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
```

Alias examples

```
prompt$ alias hi=’echo "Hello world"’
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi=’echo "Hello world"’
prompt$ 
```

Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
```

Alias examples

```
prompt$ alias hi=’echo "Hello world"’
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi=’echo "Hello world"’
prompt$ unalias hi
prompt$ █
```

Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
```

Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ █
```

Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
```

Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ █
```

Alias examples

```
prompt$ alias hi='echo "Hello world"'
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
```

Alias examples

```
prompt$ alias hi=’echo "Hello world"’
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi=’echo "Hello world"’
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ █
```

Alias examples

```
prompt$ alias hi=’echo "Hello world"’
prompt$ hi
Hello, world
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi=’echo "Hello world"’
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ ls
```

Alias examples

```
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ ls
./        bar.txt   foo.txt   Readme
../       .bashrc   hello.c   .ssh/
prompt$ 
```

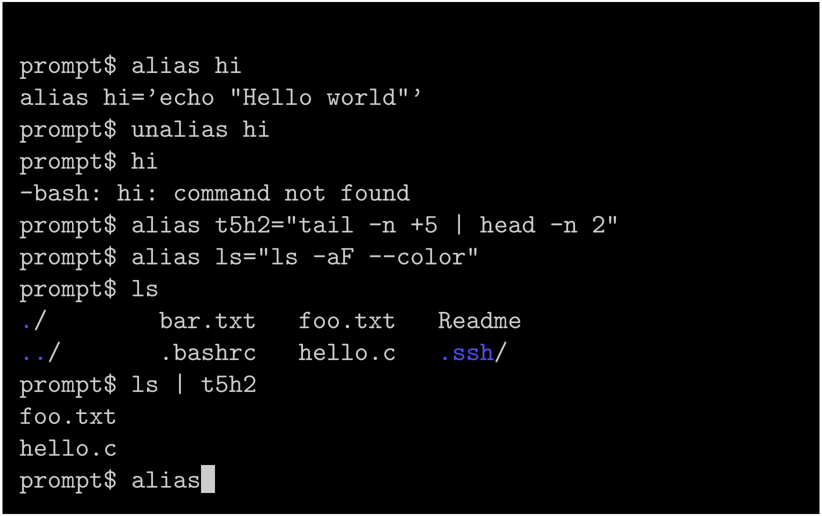
Alias examples

```
prompt$ echo hi
hi
prompt$ echo $hi

prompt$ alias hi
alias hi=’echo "Hello world"’
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ ls
./        bar.txt   foo.txt   Readme
../       .bashrc   hello.c   .ssh/
prompt$ ls | t5h2
```

Alias examples

```
prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ ls
./         bar.txt    foo.txt    Readme
../        .bashrc    hello.c    .ssh/
prompt$ ls | t5h2
foo.txt
hello.c
prompt$ █
```

Alias examples



```
prompt$ alias hi
alias hi='echo "Hello world"'
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ ls
./         bar.txt   foo.txt   Readme
../        .bashrc   hello.c   .ssh/
prompt$ ls | t5h2
foo.txt
hello.c
prompt$ alias
```

Alias examples

```
prompt$ unalias hi
prompt$ hi
-bash: hi: command not found
prompt$ alias t5h2="tail -n +5 | head -n 2"
prompt$ alias ls="ls -aF --color"
prompt$ ls
./          bar.txt   foo.txt   Readme
../         .bashrc   hello.c   .ssh/
prompt$ ls | t5h2
foo.txt
hello.c
prompt$ alias
alias ls='ls -aF --color'
alias t5h2='tail -n +5 | head -n 2'
prompt$ █
```

Evil prank

Do this next time your friend leaves a machine unattended

1. `alias ls='echo "No files."'`
2. When they get back:
   “Hey, someone came over and deleted all your files.”

Evil prank

Do this next time your friend leaves a machine unattended

1. `alias ls='echo "No files."'`
2. When they get back:
   "Hey, someone came over and deleted all your files."

Important lesson: never leave your machine unattended

- Always lock your screen
- Prevents these pranks
- Prevents much more sinister actions
- You can be fired for leaving your account unlocked

Exporting variables

- Exported shell variables are copied into children processes
  - *Any* process, not just another shell
- To export a shell variable

```
export VARNAME
```

- You can export and define a variable at the same time

```
export VARNAME=value
```

- To see a list of exported variables

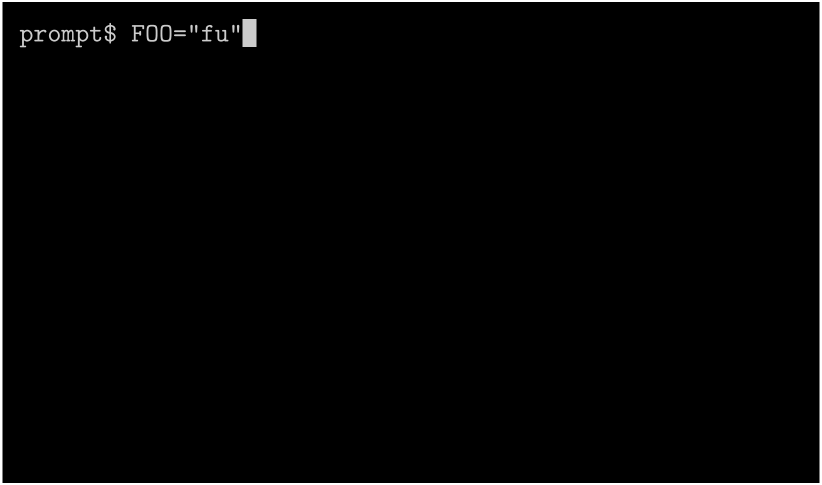
```
export
```

- By convention, exported variable names are all caps

export example

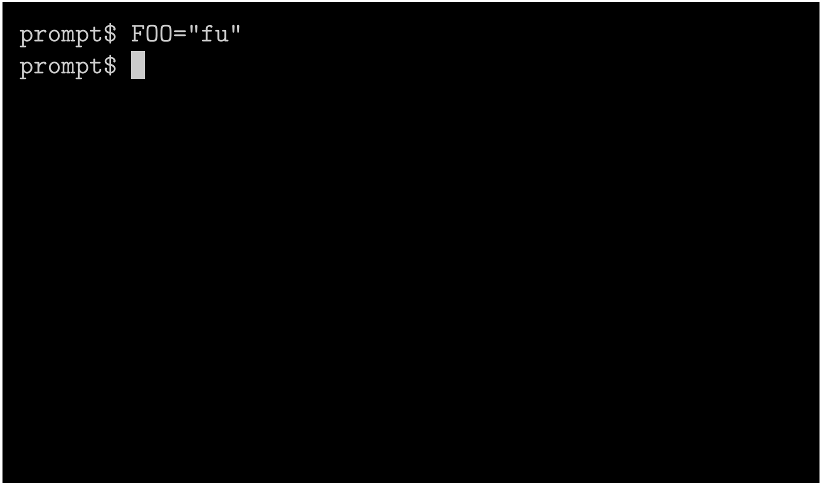
```
prompt$ █
```

export example



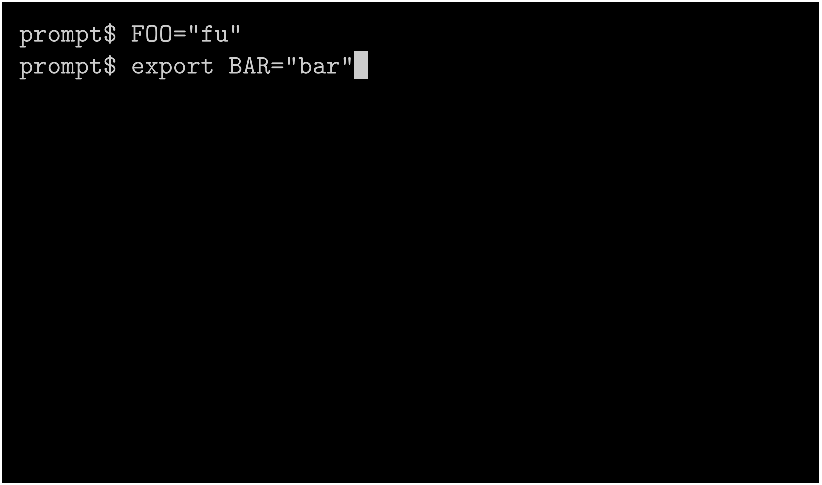
```
prompt$ FOO="fu"
```

export example



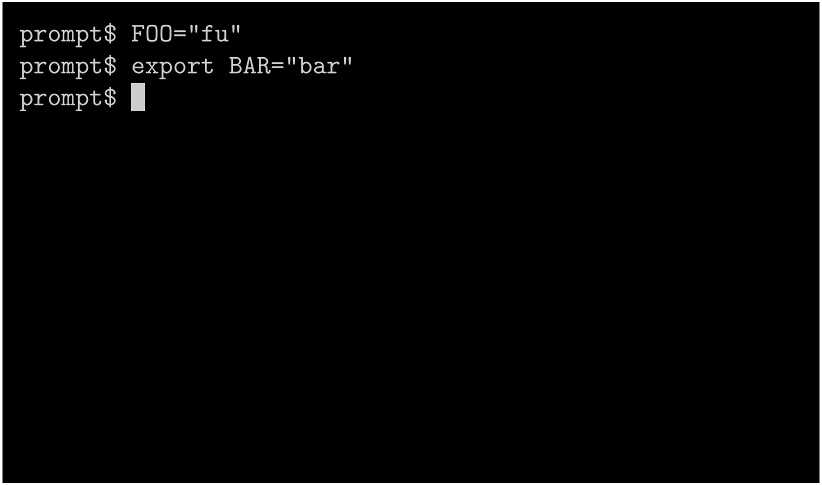
```
prompt$ FOO="fu"
prompt$ 
```

export example



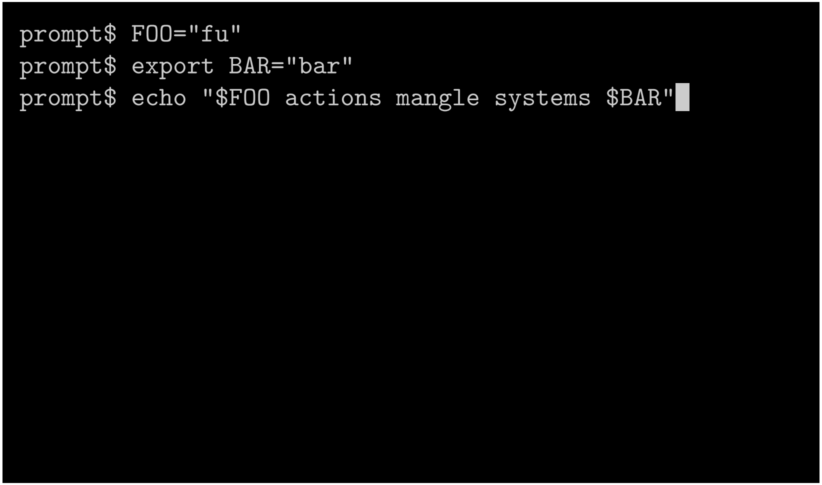
```
prompt$ FOO="fu"
prompt$ export BAR="bar"
```

export example



```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ 
```

export example

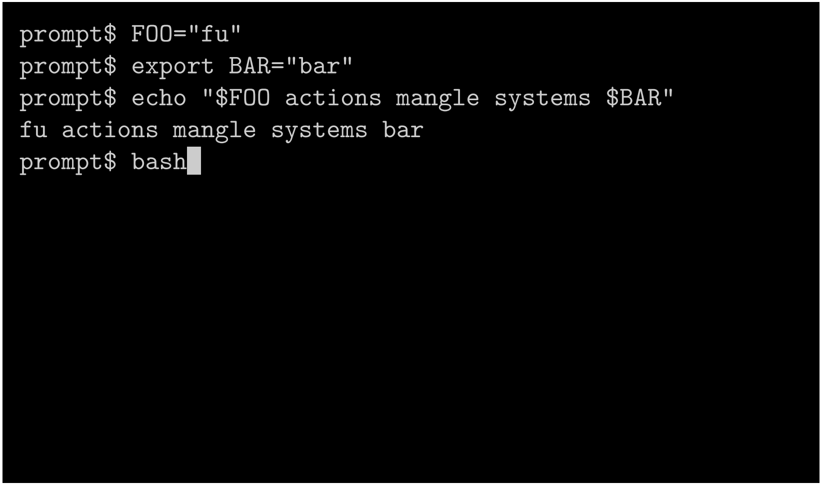


```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
```

export example

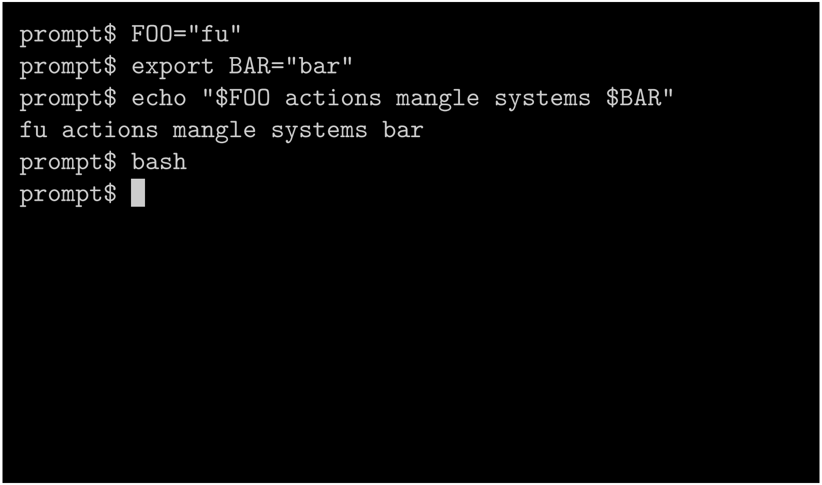
```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ █
```

export example



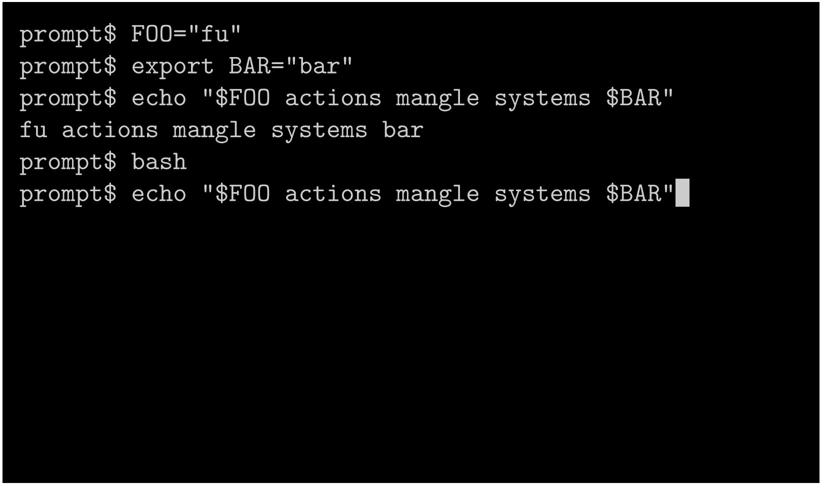
```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
```

export example



```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ 
```

export example

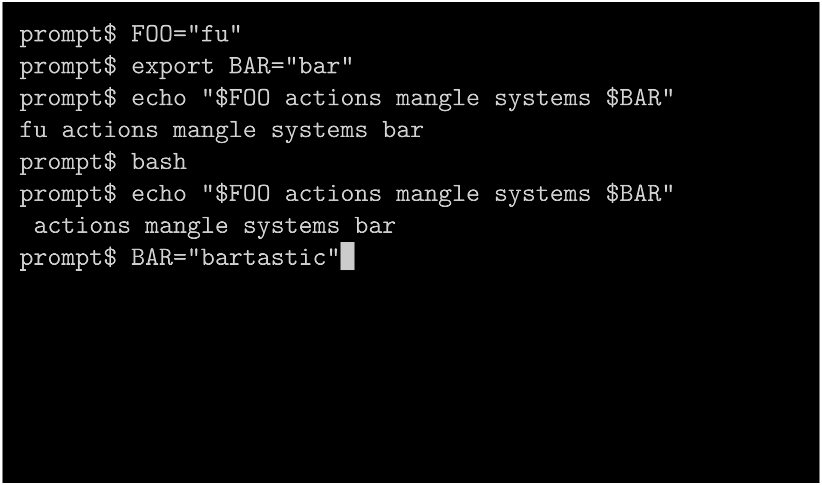


```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ █
```

export example



```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ █
```

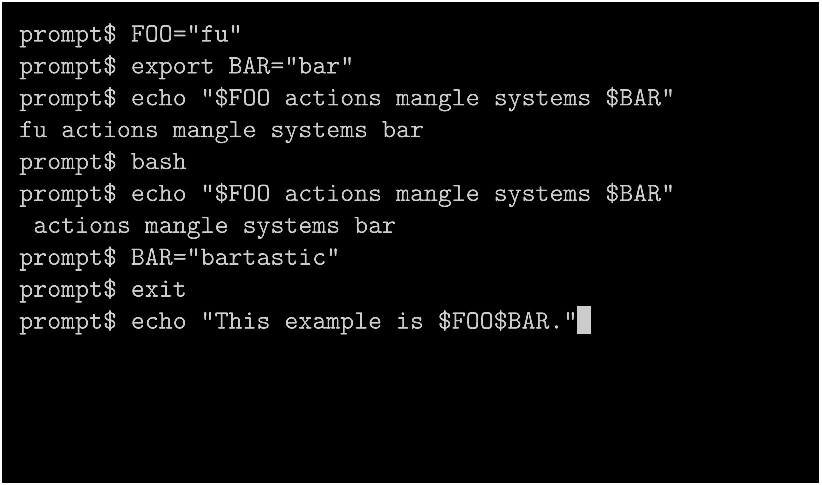
export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
prompt$ █
```

export example



```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
prompt$ echo "This example is $FOO$BAR."
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
prompt$ echo "This example is $FOO$BAR."
This example is fubar.
prompt$ 
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
prompt$ echo "This example is $FOO$BAR."
This example is fubar.
prompt$ export
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
prompt$ echo "This example is $FOO$BAR."
This example is fubar.
prompt$ export
declare -x BAR="bar"
prompt$ 
```

Environment

- The collection of exported variables is called the environment
- Can be used to adjust behavior of applications
  - Remember, any process has access to the environment
  - In C, use

```
int main(int argc, char** argv, char** env)
```

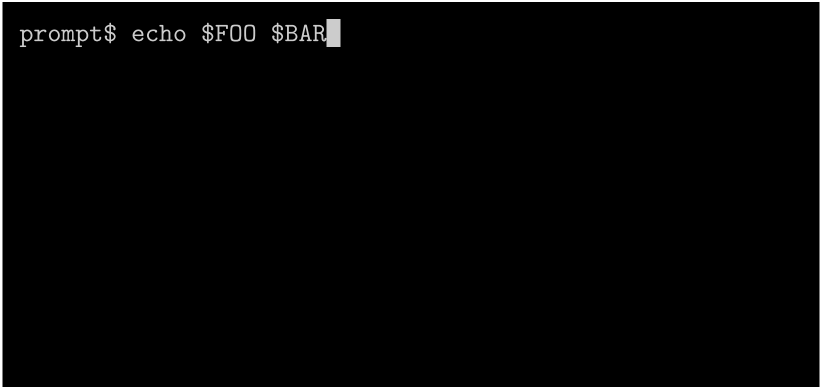
env: manage the environment

- Usage: `env [name=value] ...[name=value] [cmd args]`
- Run "`cmd args`" with an adjusted environment
- If no "`cmd`" is given, display the environment

env example

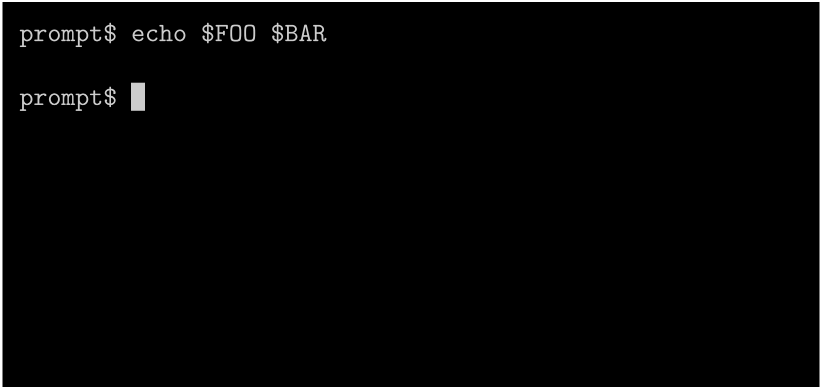
```
prompt$
```

env example



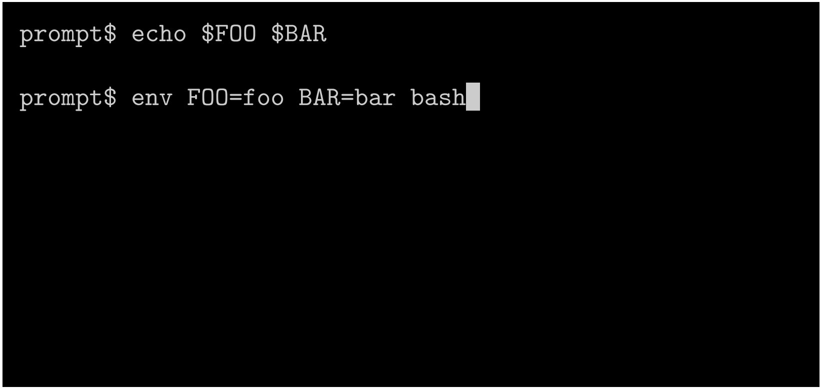
```
prompt$ echo $FOO $BAR
```

env example



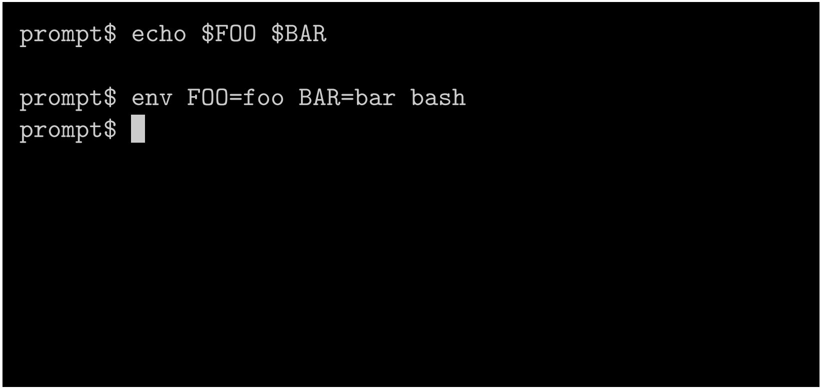
```
prompt$ echo $FOO $BAR

prompt$ 
```

env example



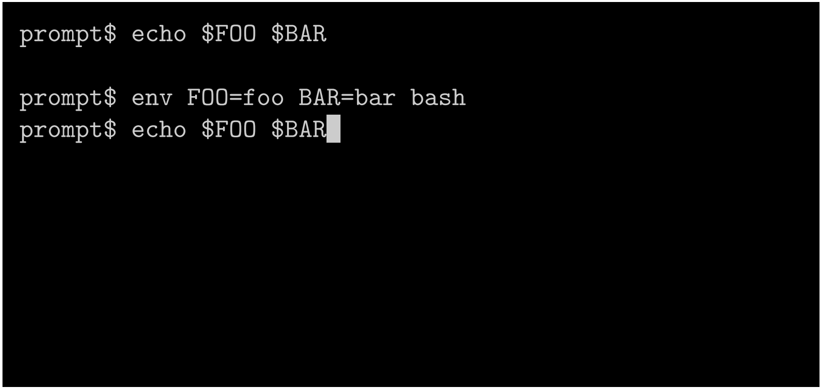
```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
```

env example



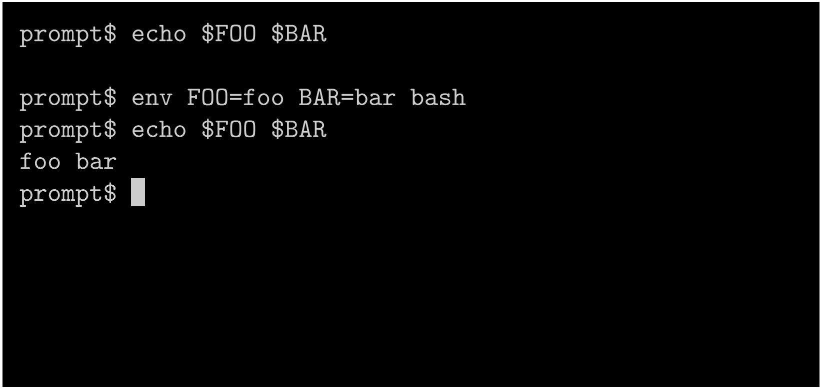
```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ 
```

env example



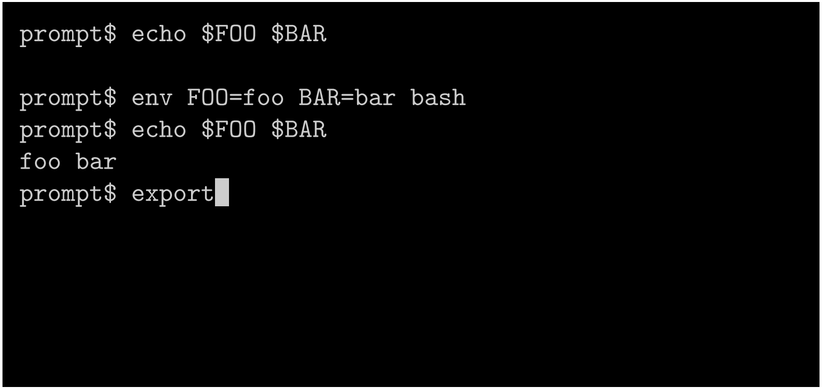
```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
```

env example



```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$
```

env example

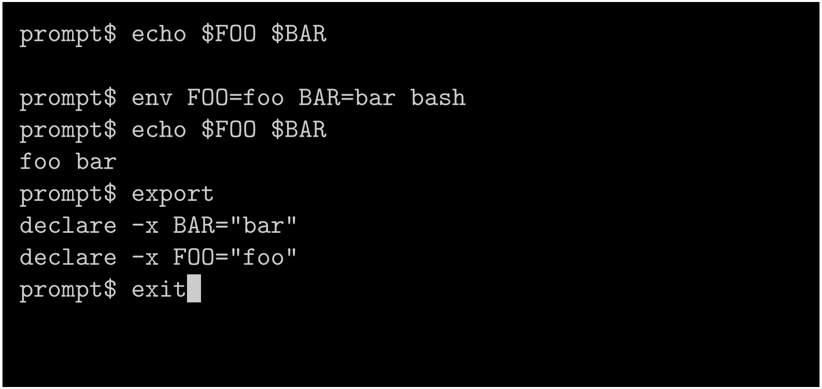


```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$ export
```

env example

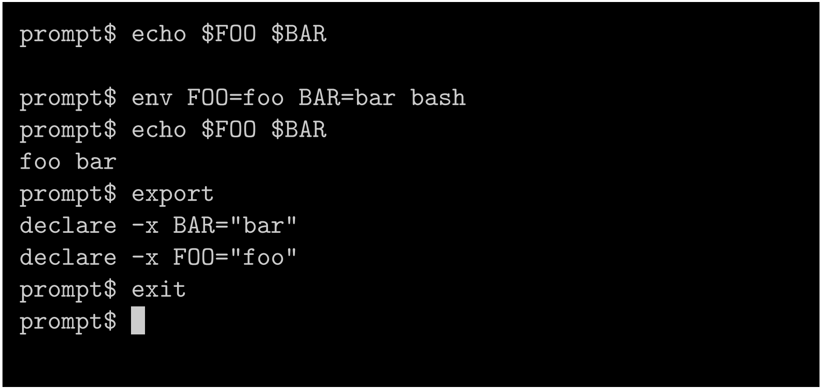
```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$ export
declare -x BAR="bar"
declare -x FOO="foo"
prompt$ 
```

env example



```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$ export
declare -x BAR="bar"
declare -x FOO="foo"
prompt$ exit
```

env example



```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$ export
declare -x BAR="bar"
declare -x FOO="foo"
prompt$ exit
prompt$ 
```

env example

```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$ export
declare -x BAR="bar"
declare -x FOO="foo"
prompt$ exit
prompt$ export
```

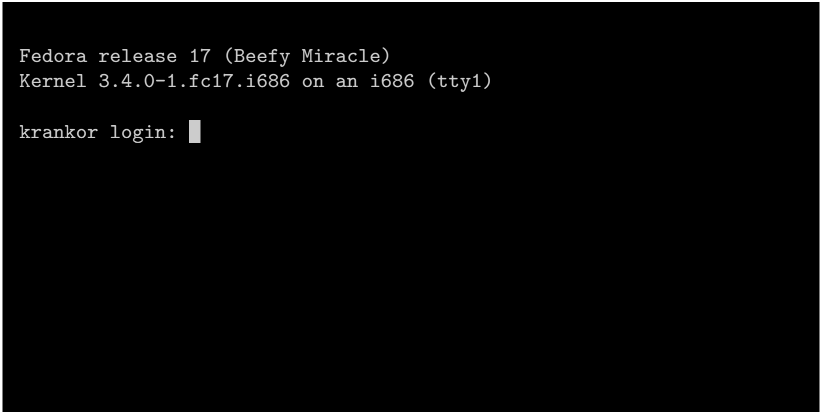
env example

```
prompt$ echo $FOO $BAR

prompt$ env FOO=foo BAR=bar bash
prompt$ echo $FOO $BAR
foo bar
prompt$ export
declare -x BAR="bar"
declare -x FOO="foo"
prompt$ exit
prompt$ export
prompt$ █
```

Reality check

- The previous examples were "convenient fiction"
  - Benefit of the canned examples
  - I will occasionally "withhold the whole truth" from you
- Your actual shell environment will have many variables defined
- But why?
  - Environment variables are used to tweak utilities
  - Saves you from having to type switches every time
  - Example: the `$PAGER` environment variable
    - Specify your favorite pager
    - Used by utilities (like `man`) that pipe things through pagers
    - May not be present — most will default to "`less`"

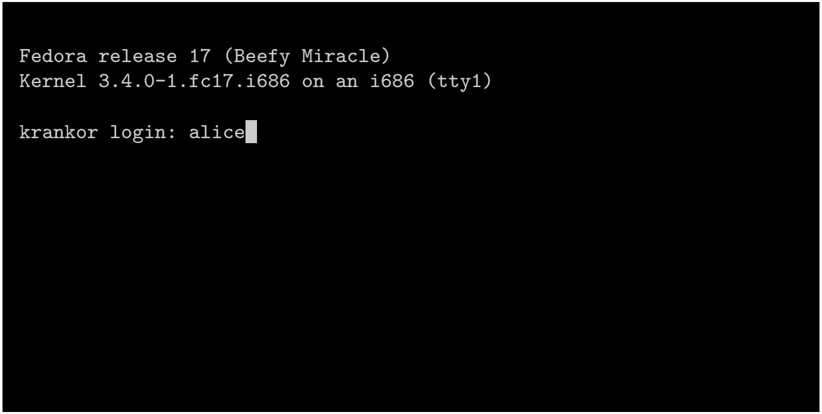
Fun with environment variables[2]



```
Fedora release 17 (Beefy Miracle)
Kernel 3.4.0-1.fc17.i686 on an i686 (tty1)

krankor login: 
```

[2] Even this is not the “whole” truth...

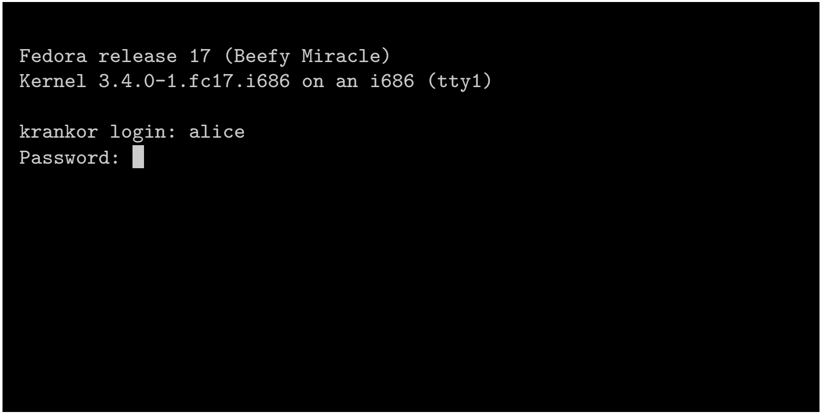
Fun with environment variables[2]



```
Fedora release 17 (Beefy Miracle)
Kernel 3.4.0-1.fc17.i686 on an i686 (tty1)

krankor login: alice
```

[2]Even this is not the “whole” truth. . .

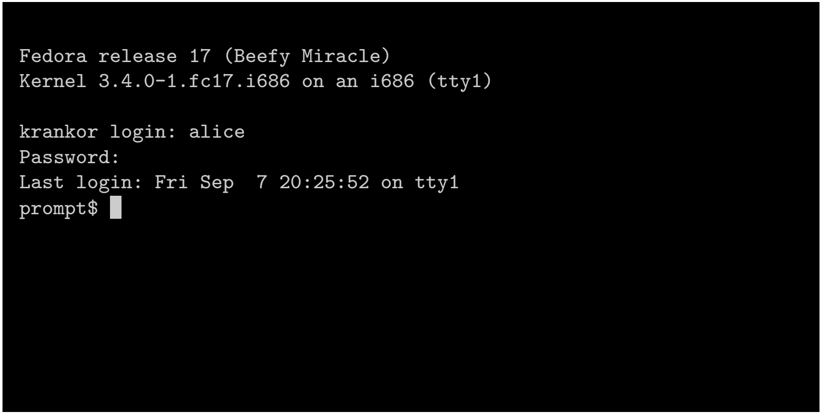
Fun with environment variables[2]



```
Fedora release 17 (Beefy Miracle)
Kernel 3.4.0-1.fc17.i686 on an i686 (tty1)

krankor login: alice
Password: 
```

[2] Even this is not the "whole" truth...

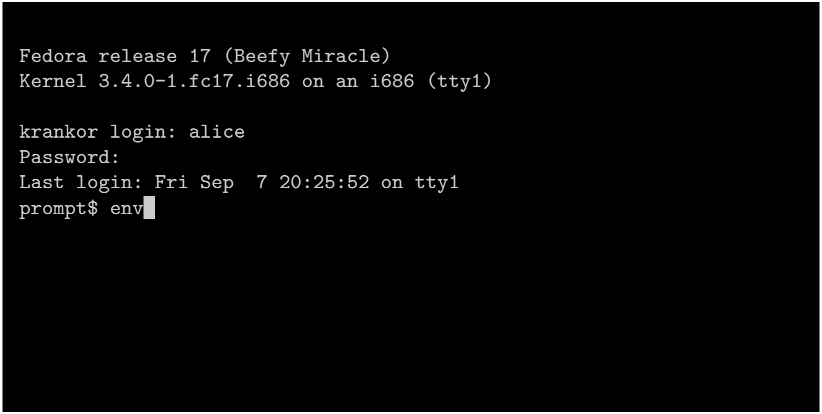
Fun with environment variables[2]



```
Fedora release 17 (Beefy Miracle)
Kernel 3.4.0-1.fc17.i686 on an i686 (tty1)

krankor login: alice
Password:
Last login: Fri Sep  7 20:25:52 on tty1
prompt$
```

[2] Even this is not the "whole" truth. . .

Fun with environment variables[2]



```
Fedora release 17 (Beefy Miracle)
Kernel 3.4.0-1.fc17.i686 on an i686 (tty1)

krankor login: alice
Password:
Last login: Fri Sep  7 20:25:52 on tty1
prompt$ env
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
Password:
Last login: Fri Sep  7 20:25:52 on tty1
prompt$ env
HOSTNAME=krankor
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ █
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
Password:
Last login: Fri Sep  7 20:25:52 on tty1
prompt$ env
HOSTNAME=krankor
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
Last login: Fri Sep  7 20:25:52 on tty1
prompt$ env
HOSTNAME=krankor
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ █
```

[2]Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
Last login: Fri Sep  7 20:25:52 on tty1
prompt$ env
HOSTNAME=krankor
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
prompt$ env
HOSTNAME=krankor
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ █
```

[2]Even this is not the "whole" truth. . .

Fun with environment variables[2]

```
prompt$ env
HOSTNAME=krankor
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
```

[2] Even this is not the “whole” truth…

Fun with environment variables[2]

```
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ █
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
TERM=linux
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ 
```

[2]Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
SHELL=/bin/bash
USER=alice
MAIL=/var/spool/mail/alice
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
```

[2] Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                   MAN(1)

prompt$
```

[2] Even this is not the “whole” truth…

Fun with environment variables[2]

```
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                  MAN(1)

prompt$ PATH=""
```

[2]Even this is not the "whole" truth. . .

Fun with environment variables[2]

```
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                    MAN(1)

prompt$ PATH=""
prompt$ █
```

[2] Even this is not the "whole" truth. . .

Fun with environment variables[2]

```
PWD=/home/alice
HOME=/home/alice
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                    MAN(1)

prompt$ PATH=""
prompt$ ls /home/alice
```

[2] Even this is not the "whole" truth...

Fun with environment variables[2]

```
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                    MAN(1)

prompt$ PATH=""
prompt$ ls /home/alice
-bash: ls: No such file or directory
prompt$ █
```

[2]Even this is not the “whole” truth. . .

Fun with environment variables[2]

```
LOGNAME=alice
PAGER=/bin/less
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                    MAN(1)

prompt$ PATH=""
prompt$ ls /home/alice
-bash: ls: No such file or directory
prompt$ /bin/ls /home/alice
```

[2]Even this is not the "whole" truth...

Fun with environment variables[2]

```
_=/bin/env
prompt$ HOME=/etc
prompt$ cd ~
prompt$ pwd
/etc
prompt$ PAGER="/bin/head -n 2"
prompt$ man man
MAN(1)                    Manual pager utils                    MAN(1)

prompt$ PATH=""
prompt$ ls /home/alice
-bash: ls: No such file or directory
prompt$ /bin/ls /home/alice
bar.txt   foo.txt   hello.c   README
prompt$ 
```

[2] Even this is not the "whole" truth...

Your `PATH`

Remember: there are two types of "commands"

1. Shell "builtins": managed by the shell
2. Executables that live somewhere (like, `ls`)

Your PATH

Remember: there are two types of “commands”

1. Shell “builtins”: managed by the shell
2. Executables that live somewhere (like, `ls`)

What does the shell do when the command is not built–in?

Your PATH

Remember: there are two types of “commands”

1. Shell “builtins”: managed by the shell
2. Executables that live somewhere (like, `ls`)

What does the shell do when the command is not built–in?

1. Find and run the executable
2. Print an error if the executable is not found

Your PATH

Remember: there are two types of "commands"

1. Shell "builtins": managed by the shell
2. Executables that live somewhere (like, `ls`)

What does the shell do when the command is not built–in?

1. Find and run the executable
2. Print an error if the executable is not found

How does the shell know where to look for an executable?

Your `PATH`

Remember: there are two types of "commands"

1. Shell "builtins": managed by the shell
2. Executables that live somewhere (like, `ls`)

What does the shell do when the command is not built–in?

1. Find and run the executable
2. Print an error if the executable is not found

How does the shell know where to look for an executable?

If you specify a directory, the shell just tries that one. Otherwise:

Your `PATH`

Remember: there are two types of "commands"

1. Shell "builtins": managed by the shell
2. Executables that live somewhere (like, `ls`)

What does the shell do when the command is not built–in?

1. Find and run the executable
2. Print an error if the executable is not found

How does the shell know where to look for an executable?

If you specify a directory, the shell just tries that one. Otherwise:

- The `PATH` variable specifies a list of directories
- The shell checks those directories in order
- First matching executable is the one that runs

Your PATH

Remember: there are two types of "commands"

1. Shell "builtins": managed by the shell
2. Executables that live somewhere (like, `ls`)

What does the shell do when the command is not built–in?

1. Find and run the executable
2. Print an error if the executable is not found

How does the shell know where to look for an executable?

If you specify a directory, the shell just tries that one. Otherwise:

- The `PATH` variable specifies a list of directories
- The shell checks those directories in order
- First matching executable is the one that runs

Be careful with your `PATH`. More on this later...

How can I set things for *every* shell I start

How can I set things for *every* shell I start

Easy — edit your `.bashrc` file

- In your home directory
- Sometimes has another name, so check your `man` pages
- Is an ordinary text file
- The file contains `bash` "commands"
  - Anything you could type in a `bash` shell
- This file is executed whenever you start a shell
  - There are switches to change this...

Sample .bashrc file (well, part of one)

```
prompt$ █
```

Sample .bashrc file (well, part of one)

```
prompt$ cat .bashrc
```

Sample .bashrc file (well, part of one)

```
umask 077

alias cp='cp -i'
alias mv='mv -i'
alias rm='rm -i'
alias ls='ls -aF --color'
alias man='man -a'

export SVN_EDITOR="/usr/bin/vim"

export PATH=$PATH:~/bin

prompt$
```

Arithmetic in bash

expr: evaluate an expression

- The expression is passed as arguments
- Each term of the expression must be its own argument
  - Be sure to leave spaces everywhere
- Allowed operators:

  +, -, *, /, %, |, &,

  <, <=, >, >=, =, !=
- Can use parentheses to group
- Check your man pages for more details
- Be careful with shell special characters

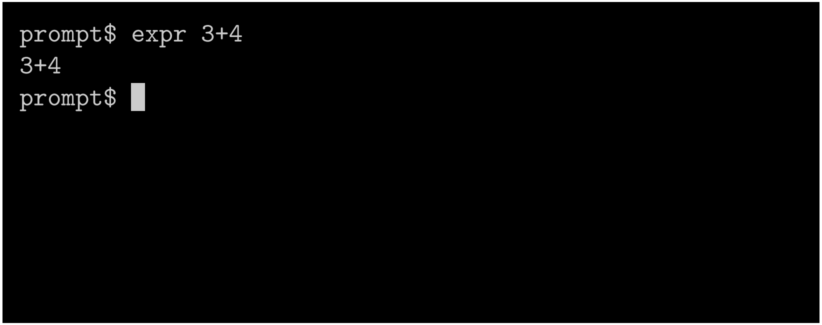
expr examples

```
prompt$
```

expr examples

```
prompt$ expr 3+4
```

expr examples



```
prompt$ expr 3+4
3+4
prompt$ 
```

What happened?

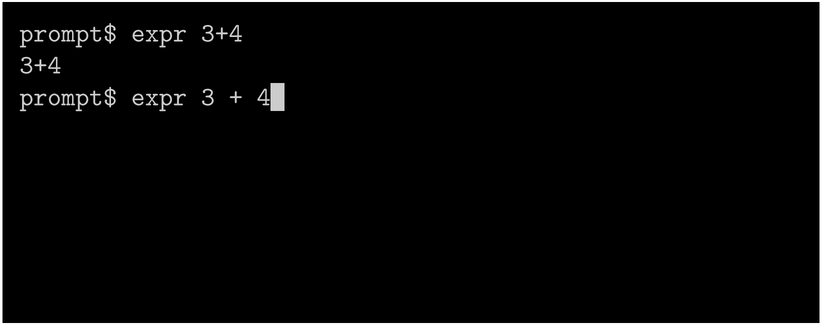
expr examples

```
prompt$ expr 3+4
3+4
prompt$ █
```

What happened?

Need spaces between terms

expr examples



```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
```

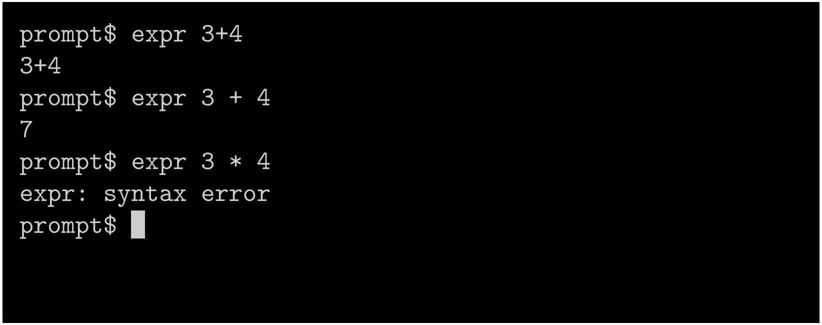
expr examples

```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ 
```

expr examples

```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
```

expr examples



```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$
```

What happened?

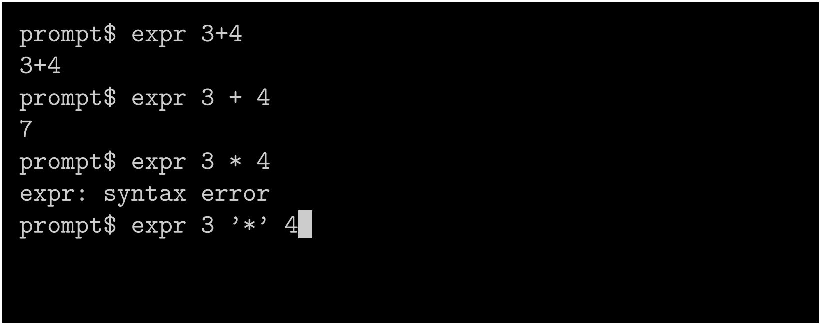
expr examples

```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ █
```

What happened?

"*" is replaced by files in working directory

expr examples



```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
```

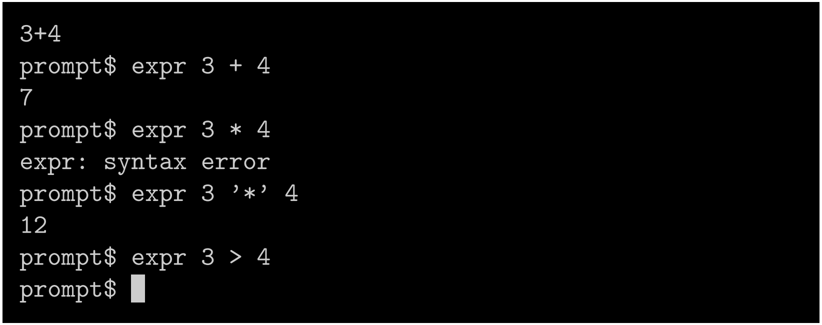
expr examples

```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ 
```

expr examples

```
prompt$ expr 3+4
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ expr 3 > 4
```

expr examples



```
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ expr 3 > 4
prompt$
```

What happened?

expr examples

```
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ expr 3 > 4
prompt$ 
```

What happened?

">" is for redirection

expr examples

```
3+4
prompt$ expr 3 + 4
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ expr 3 > 4
prompt$ cat 4
```

expr examples

```
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 ’*’ 4
12
prompt$ expr 3 > 4
prompt$ cat 4
3
prompt$ █
```

expr examples

```
7
prompt$ expr 3 * 4
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ expr 3 > 4
prompt$ cat 4
3
prompt$ expr 3 \> 4
```

expr examples

```
expr: syntax error
prompt$ expr 3 '*' 4
12
prompt$ expr 3 > 4
prompt$ cat 4
3
prompt$ expr 3 \> 4
0
prompt$ █
```

Time for a quiz

- Suppose I have a shell variable, `x`
- Suppose I know it contains an integer (as a string)
- How can I increment `x`?

Time for a quiz

- Suppose I have a shell variable, `x`
- Suppose I know it contains an integer (as a string)
- How can I increment `x`?
- Using “backward quotes”:

```
x=`expr $x + 1`
```

- Using “`$(...)`”:

```
x=$(expr $x + 1)
```

Shorthand for arithmetic

Good news

```
$(expr arg1 arg2 ...  argn)
```

can be instead written as

```
$[arg1 arg2 ...  argn]
```

and the `$[...]` environment is a little nicer:

- Unnecessary to leave spaces
- Unnecessary to escape the shell special characters
- We can drop the “$” when referring to variables

Shorthand vs. longhand example

The following are equivalent:

```
y=$(expr '(' $x + 1 ')' '*' 2)
```

```
y=`expr '(' $x + 1 ')' '*' 2`
```

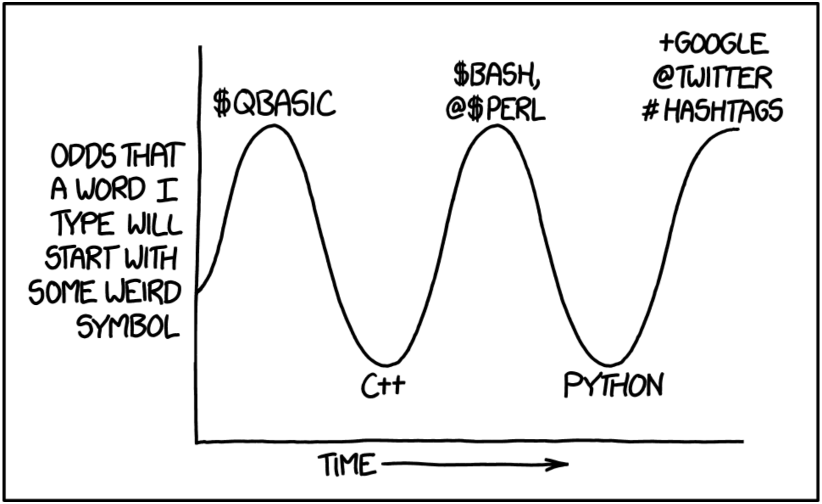
```
y=$[ ( $x + 1 ) * 2 ]
```

```
y=$[ ( x + 1 ) * 2 ]
```

```
y=$[(x+1)*2]
```

alias	manage aliases
env	manage environment
export	export shell variables
expr	evaluate an expression
read	read from standard input, into variables
unalias	remove an alias

An appropriate xkcd comic: `http://xkcd.com/1306`


ODDS THAT A WORD I TYPE WILL START WITH SOME WEIRD SYMBOL
$QBASIC
C++
$BASH, @$PERL
PYTHON
+GOOGLE @TWITTER #HASHTAGS
TIME

End of lecture