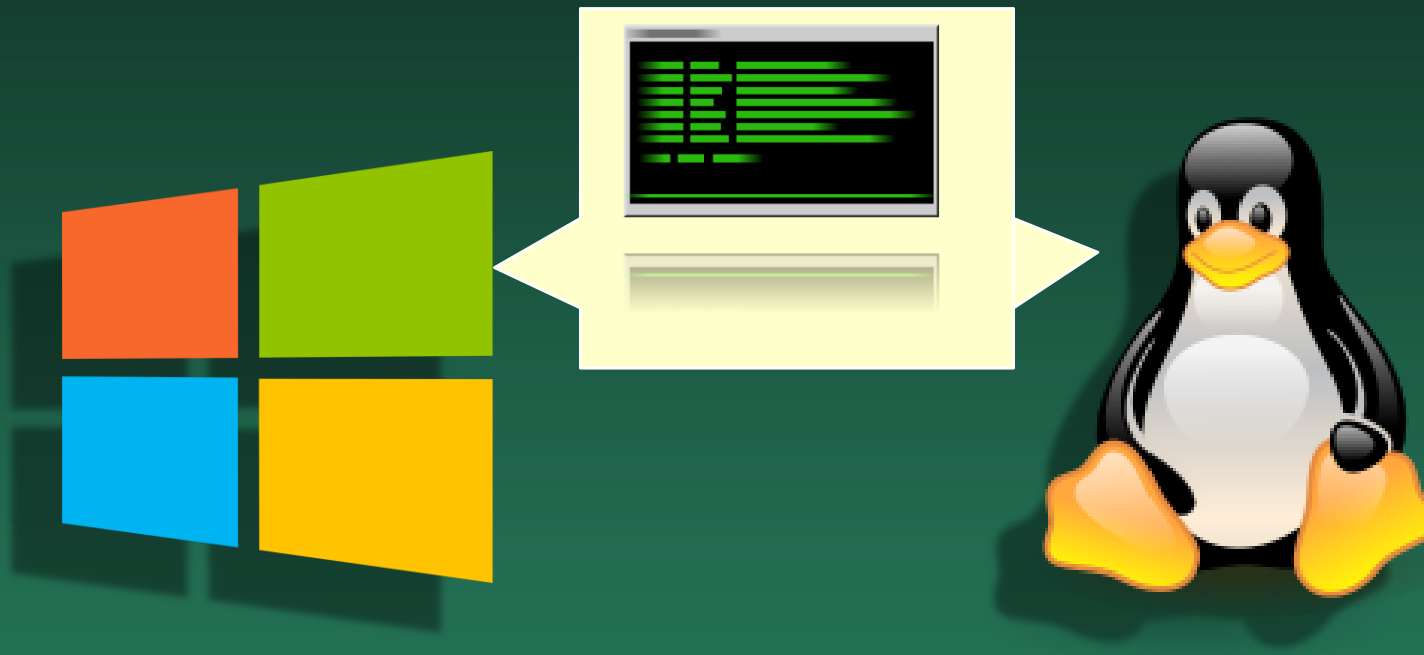
System shells



BASH syntax - 8

3 - BASH Scripts

#!/bin/bash

BASH script file

- Type a group of commands in sh01

```
cd ${HOME}
mkdir LABS
cd LABS
mkdir AA BB CC
date -I > AA\dd.txt
date +%I:%M:%S\ %p > BB\tt.txt
echo $NAME > CC\uu.txt
cat */*.txt > here.txt
cd ..
cat LABS/here.txt
read -p "press <ENTER> to cleanup"
rm -r LABS
```

- Run the commands
  - `bash sh01` ...does the job
  - `./sh01` ...works too
  - `sh sh01` ...again,

BASH file format

- First line in script file

```
#!/bin/bash

echo "The shell is ${SHELL}"
```

Shebang
Runtime information

  - Compatibility
    - `#!/bin/sh`
    - `#!/bin/ksh`
    - `#!/bin/csh`
  - Documentation, Tools

- No specific extension
  - `shebang` does the job
  - No extension → acts just like a command
  - `.sh` extension → avoid confusion with *Executables*
    - *from now on we'll apply the .sh extension*

Script special variables

- Type another group of commands

```
#!/bin/bash

echo $0
echo $1-$2-$3-$4-$5
echo $6-$7-$8-$9-${10}
```

- Run script
  - Shows script arguments
  - `$0` is the script name
  - `$1-etc.` are the following words
  - `$#` is the number of arguments *(not counting `$0`)*
  - `$@` is the serialization of all arguments
- `shift` will change parameters indexes
  - May take a number as an offset (e.g. `shift 2`)

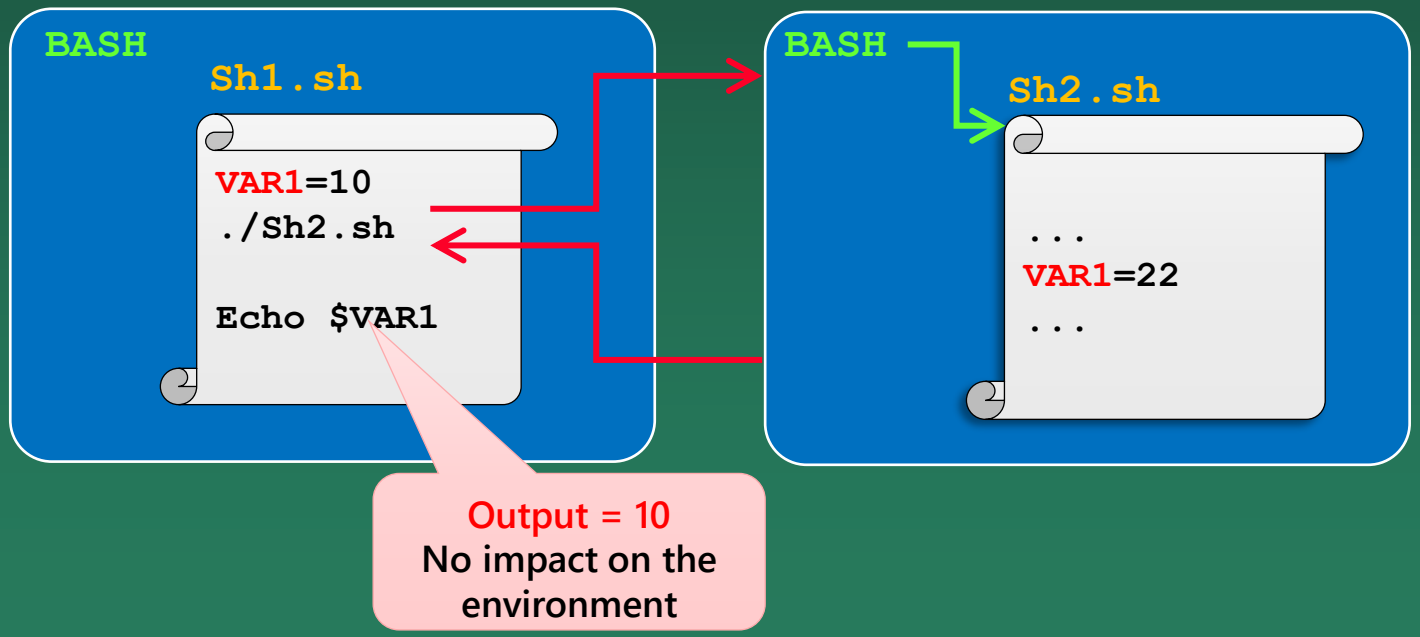
Chain your scripts

- Direct call

```
#!/bin/bash

./bsh06.sh AA BB CC
bash bsh06.sh DD EE FF
```

- The script will execute in its own environment
  - No impact on caller



Isolate lines in a script

- Type a group of commands

```
#!/bin/bash

read -p "... Type a word " ANSWER

( read -p "...another Word " ANSWER )

echo "You typed $ANSWER"
```

- Run script
  - The (...) runs in a separate shell
  - Does not impact variables in the original shell

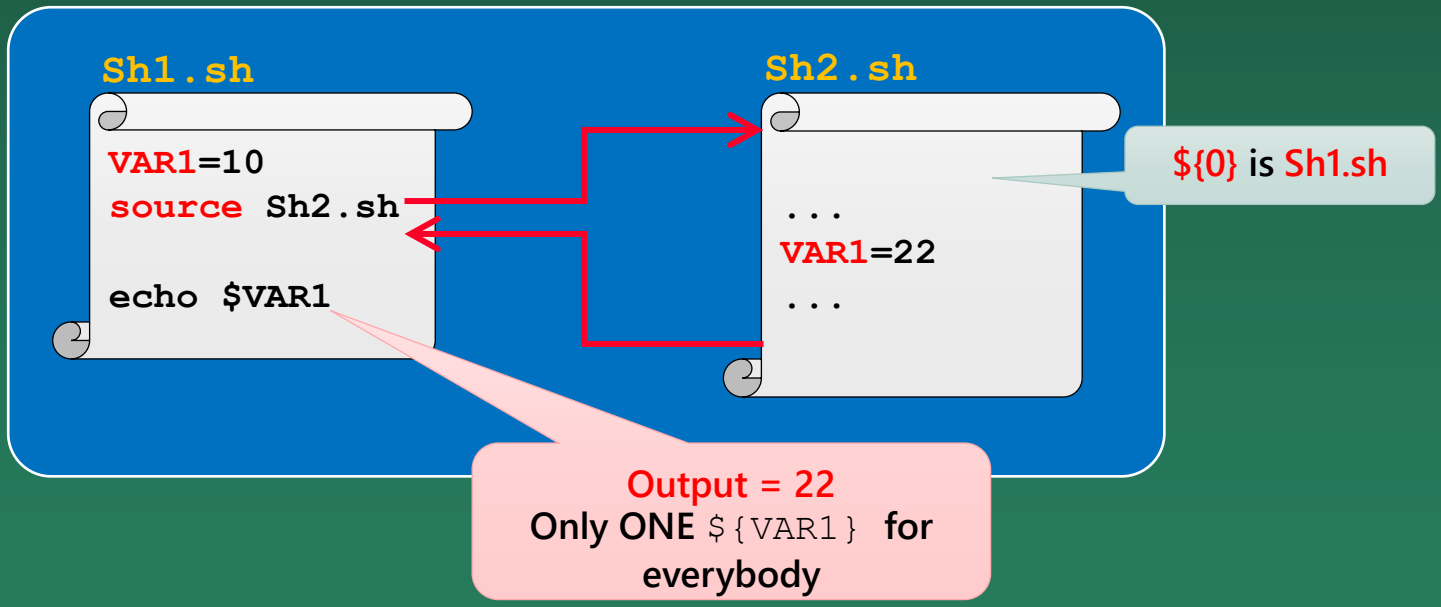
Connect your scripts

- Call with `source`

```
#!/bin/bash

source bsh06.sh II JJ KK
```

- Variables are shared
  - Calling a script may create variables collisions
  - NO collisions on `$1, $2 etc.`



Bash functions

- Bash code

```
#!/bin/bash#
func1() {
  echo "$VAR1 ... set to $1"
  VAR1=$1
}
func2() {
  echo "$VAR1 ... set to $1"
  local VAR1=$1
}
VAR1=AA
func1 DDD
echo "var1 is now $VAR1"
func2 ZZZ
echo "var1 is now $VAR1"
```

- Variables have a global scope
  - Unless they are declared `local` in the function