COMP2101
Summer 2022

bash Data Handling

©DENNIS SIMPSON 2013-2022

Working With Data

- Data on the command line
- Variables
- Dynamic data
- User input

©DENNIS SIMPSON 2013-2022

Bash Data Handling

- Bash generally treats command line data as text, in keeping with UNIX interoperability philosophy
- Bash can use any system utility to produce data
- Data produced is usually displayed and discarded by the shell
- Bash provides variables as a mechanism to temporarily store simple data

©DENNIS SIMPSON 2013-2022

Command Line Data

- Bash command lines are evaluated by bash before being executed as commands
- Bash evaluates lines using special characters such as quotes, escapes, substitutions, and arithmetic
- Bash then splits a command line into words (aka tokens), with the first word being the command to run and the rest being things for that command to work on (referred to as arguments)
- You can use the echo command to display simple text such as a label or title, or evaluated text, from doing variable data lookup, arithmetic, or command execution

```
echo "It's fun!"
echo A megabyte is $(( 1024 * 1024 )) bytes
echo Today is `date +%A`
```

©DENNIS SIMPSON 2013-2022

Shell Data - Text

- Commands can look for text on the command line to use, each word on the line is normally treated as a separate thing to work on
- You can tell bash to ignore special characters when you need to
- You might want one or more spaces as part of a filename, or to use other special characters as plain text, such as using an apostrophe as part of a word

' ' turn off all special characters except '
" " turn off most special characters, $ () and ` is still special inside ""
\ turns off any special meanings of only the very next character after the \

```
cd
touch My File
touch 'My File'
touch "It's My File"
touch Another\ File
ls
```

©DENNIS SIMPSON 2013-2022

File Name Globbing

- As part of substitution, bash looks for *, ?, and [ ] in words on the command line
- If one or more of these are found in words on the command line, bash may try to turn those words into filenames - this is called globbing
- Give extra thought to escaping these characters on your command line

```
ls *
echo *
ls .?
```

©DENNIS SIMPSON 2013-2022

Shell Data - Numbers

- Bash can use text having only digits and a + or - sign as signed integers, if you tell it to do that - test it before using it for very large numbers
- Bash can do basic arithmetic +, -, *, /, % on integers by putting arithmetic statements inside $(( )) - there are additional operators, see ARITHMETIC EVALUATION on the bash manual page
- The (()) syntax turns off file name globbing for * inside the (())

```
echo 3 + 4
echo $(( 3 + 4 ))
echo $(( 3.6 * 1.7 ))
echo 7 divided by 2 is $(( 7 / 2 )) with a remainder of $(( 7 % 2 ))
echo Rolling dice ... $(( $RANDOM % 6 + 1 )), $(( RANDOM % 6 + 1))
```

©DENNIS SIMPSON 2013-2022

Working With Strings

- Many command line tools are available to parse and manipulate strings
- grep (used to search for patterns in text)
- tr (used to do trivial character substitutions in text)
- cut (used to extract portions of text from input text data)
- sed, awk - advanced text manipulation tools
- expr index string searchtext will return a non-zero index value if searchtext is in string
- See http://tldp.org/LDP/abs/html/string-manipulation.html for more pure bash string manipulation techniques and information

©DENNIS SIMPSON 2013-2022

Temporary Data Storage

https://indianajones.fandom.com/wiki/Hangar_51?file=400.jpg

- It is very common to have to put several pieces of data together to achieve a useful result
- There may be a small or large number of data items to work with
- They may not be generated at the same times or in the same ways
- A method is required to provide temporary storage for data that will be needed in future commands

©DENNIS SIMPSON 2013-2022

Variables

- Every process has memory-based storage to hold named data
- A named data item is called a variable
- Variables can hold any data as their value, but usually hold text data, aka strings
- Variables are created by assigning data to them, using =
- It is important to not leave any spaces around the = sign
- The assigned text must be a single token for bash, escape your spaces!

```
myvar=3
variable2="string data"
vowels=a e i o u
```

©DENNIS SIMPSON 2013-2022

Variables

- Since a variable is stored in process memory, it stays around until we get rid of it, or the process ends
- To access the data stored in a variable, use $variablename
- Non-trivial variable names must be surrounded by { }, ordinary names do not require them
- Putting # at the start of a variable name tells bash you want the count of characters in the variable, not the text from the variable

```
myvar=hello
echo $myvar
echo ${#myvar}
```

```
var[2]=silly name
echo $var[2]
echo #var[2]
```

```
var[2]="silly name"
echo ${var[2]}
echo ${#var[2]}
```

©DENNIS SIMPSON 2013-2022

Dynamic Data on the Command Line

- Most commands we run interactively use static data - data which has a fixed value that we enter literally when we type the command
- Sometimes you want to run a command using command line data that may not be a fixed value - this is called using dynamic data
- Getting data from a variable to put on the command line is a way to put dynamic data on the command line
- Bash can run a sub-shell to execute one or more commands and put the output from the sub-shell onto the command line of another command

```
today=$(date +%A)
echo "Today is $today."
```

```
todaymessage="Today is $(date +%A)."
```

©DENNIS SIMPSON 2013-2022

User Input

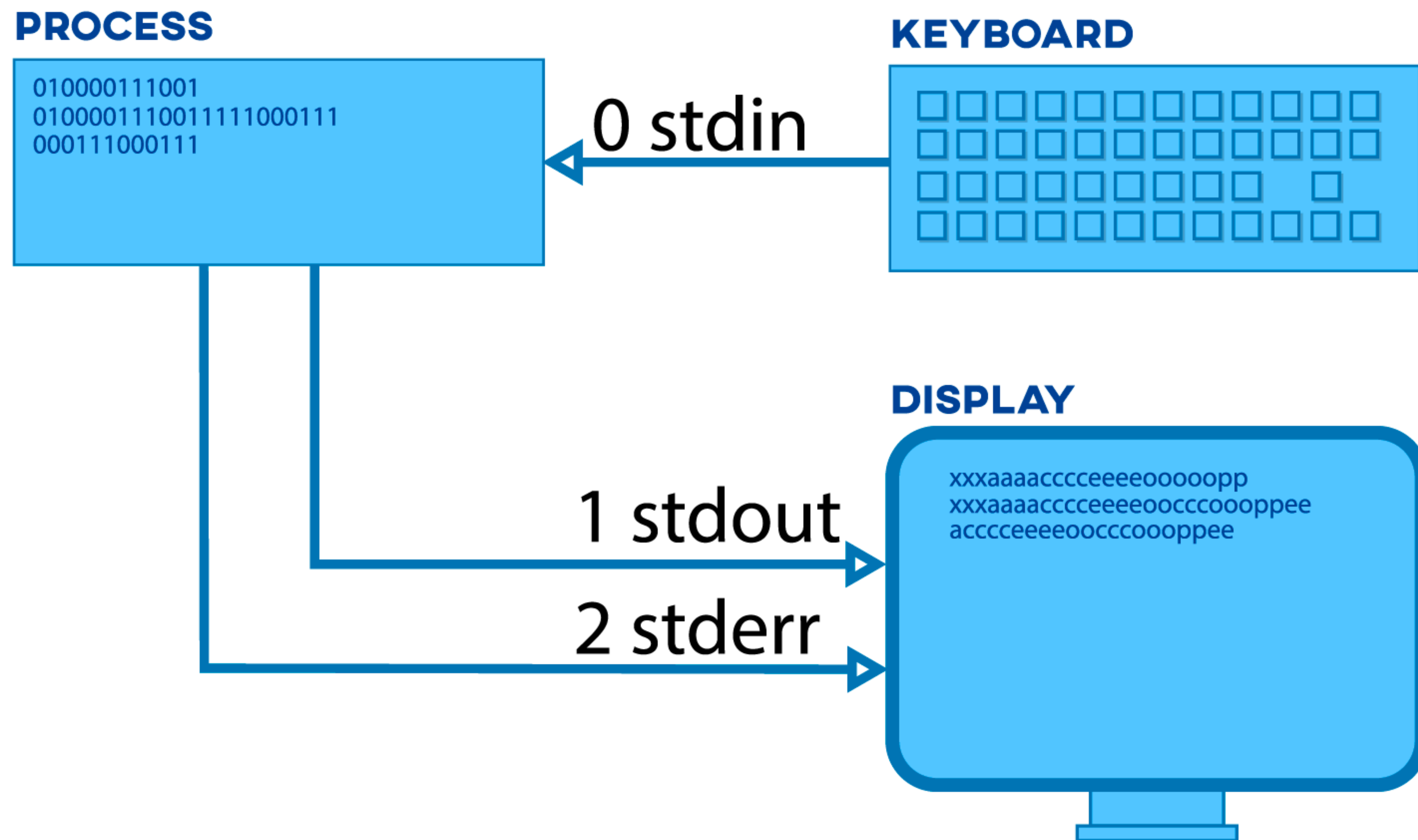
- Bash can get user input and put it in a variable
- The read command will wait for the user to type out a line of text and press enter, then put whatever was typed into a predefined variable named REPLY
- You can specify a prompt, specify what variable or variables to put the user data into, and there are other options

```
read -p "Input? " myvar
echo $myvar
```

```
prompt="Enter 2 numbers "
read -p "$prompt" usernumber1 usernumber2
echo "User gave us $usernumber1 and $usernumber2"
```

©DENNIS SIMPSON 2013-2022

Persistent Data Storage



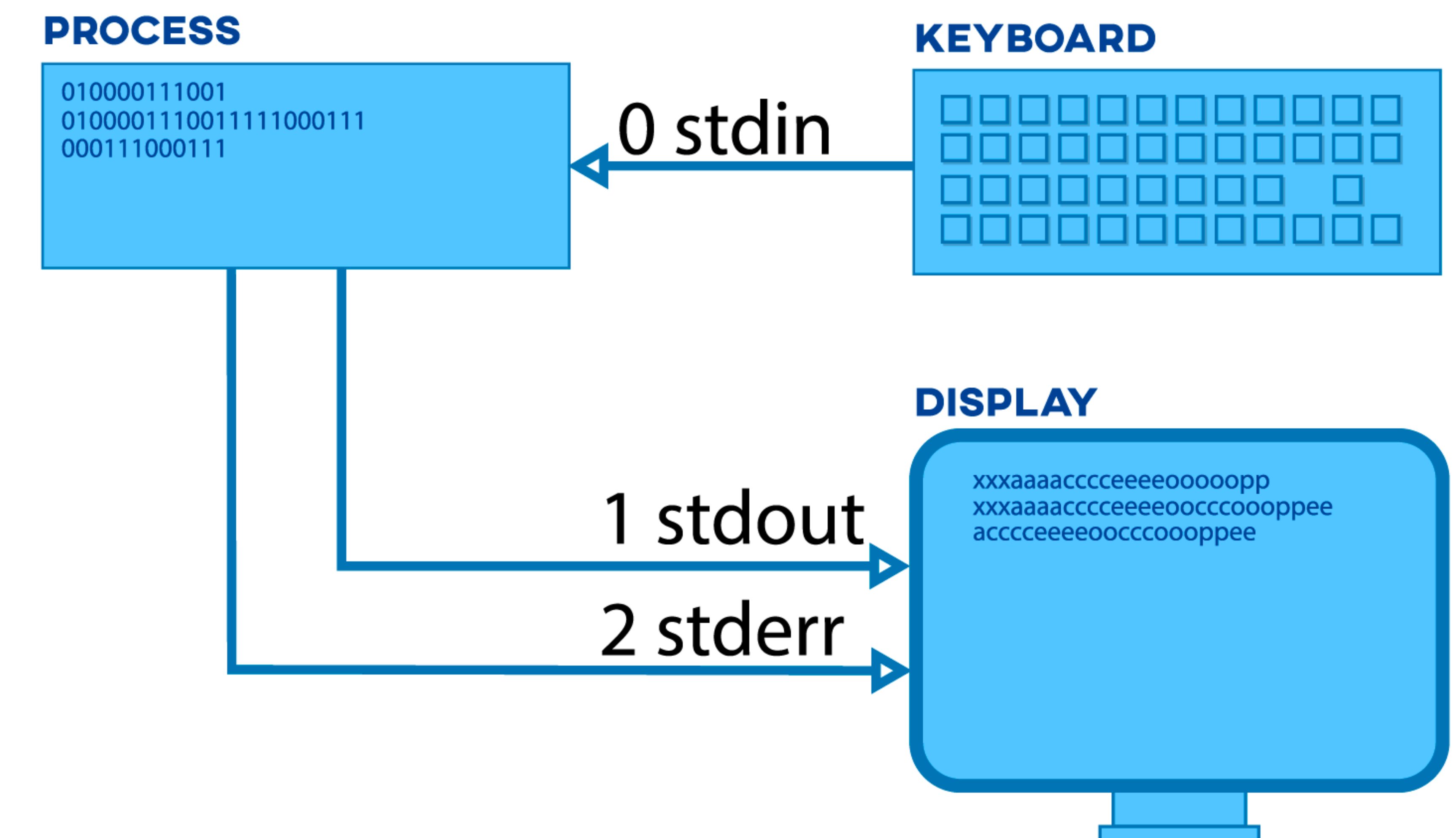
https://linuxhint.com/redirect-stderr-stdout-bash/

- Linux standard input and output
- Saving output in files
- Overwriting versus appending
- Error output redirection
- Input redirection

©DENNIS SIMPSON 2013-2022

Linux Standard Input/Output

- Processes started by bash (such as scripts) inherit one input and two output locations
- The input is called standard in, abbreviated stdin and given file descriptor 0, and is by default attached to the user's keyboard
- The two outputs are called standard out and standard error, abbreviated stdout and stderr and given file descriptors 1 and 2 respectively, are by default attached to the user's screen/window
- Running processes are expected to get user input from stdin by default
- Running processes are expected to send user output to stdout by default
- Running processes are expected to send error information to stderr by default
- All of these are changeable from their defaults when running commands from the bash command line



https://linuxhint.com/redirect-stderr-stdout-bash/

©DENNIS SIMPSON 2013-2022

Redirecting Standard Input/Output/Error

- bash can be told to redirect input and output in a number of ways
- Pipes connect stdout from one process to stdin on another process
- stdin, stdout, and stderr are entries 0, 1, and 2 in the process' file descriptor table
- A physical keyboard has a filename associated with it, as does a display screen, and those filenames are what is found in the file descriptor table for entries 0, 1, and 2 by default
- Any file can be put into the file descriptor table in those slots and they will be used for stdin, stdout, and stderr
- To set a non-default filename into a file descriptor before bash starts a process running, the > symbol is used

©DENNIS SIMPSON 2013-2022

Output Redirection

- To use > to redirect output sent to stdout by a command, put >pathname on the command line
- Prior to starting the command running, bash will temporarily change stdout for that command's process to that pathname
- The command can just make output the same way it always does, and the output will go to pathname instead of the user's display without the process caring that this is happening
- When stdout is redirected this way it only goes to the output pathname and does not come out on the screen, so if you redirect to /dev/null, output is silently discarded
- If pathname does not exist, bash will create it
- If pathname exists and has data in it already, bash will remove all data from pathname before starting the command running
- Once the command has started running, the bash process that started it will ensure its own stdout is attached to the screen so that the next bash prompt will be visible to the user

©DENNIS SIMPSON 2013-2022

Stderr Redirection and Appending

- If the error information produced by a process is what should be sent to a file instead of the normal output, use 2>pathname on the command line
- To append to a file (write output there without removing whatever was there first), use >>pathname or 2>>pathname
- To discard error output from a command, use 2>/dev/null
- stdin can also be redirected by using <pathname , <<MARKERTEXT , and <<<"literaltext"
- The file descriptor table can have many more than just 3 files in it, and those other entries can be used to do some fancy things

©DENNIS SIMPSON 2013-2022

Using Output Templates

- Output from a script can take the form of summaries or reports that have a stable format and organization
- For those situations it improve consistency and quality of output if the fixed portions of the report such as titles, labels, and spacing/delimiters are placed in their final form easily viewed and modified in the script
- Variables can be used to provide the data portions of the reports without affecting the clarity of the template

```
# this script demonstrates how to
# use a template for output

# gather my data for my report
dat1=56
dat2=$(getmydata)
dat3=$something

# print out the report using the gathered data
cat <<EOF
My Report
=========
Data 1: $dat1
Data 2: $dat2
Data 3: $dat3
=========
EOF
```

©DENNIS SIMPSON 2013-2022

Scripting Environment

https://www.therpf.com/forums/threads/star-trek-captain-kirks-bamboo-cannon-%E2%80%94-cutaway-build.267508/

- The process environment
- Shell environment files
- Environment variables

©DENNIS SIMPSON 2013-2022

Predefined Variables

- bash provides a number of variables containing useful information, the env command will display them
- Some are inherited from ancestor processes (e.g. login, getty, init)
- More are often set up by user environment files to customize the user experience
- bash also has some special variables that dynamically retrieve data for realtime uses
- Script files that contain variable assignments can be imported into your script process by sourcing those files to provide information to your script from the system

```
source ~/.myvars
source /etc/os-release
source /etc/lsb-release
```

©DENNIS SIMPSON 2013-2022

Special Variables

- $ - current process id
- # - number of parameters on the command line of a script
- 0-n - command line parameters
- RANDOM - a random 15-bit integer
- REPLY - default variable name for the read command
- ? - exit status of most recent command, zero means success
- See the bash man page for a full list of special variables

©DENNIS SIMPSON 2013-2022

Shell Process

- The bash shell is just a program stored in the file /bin/bash
- When you tell the operating system to run that program, it creates an instance of the program known as a process
- A process is a chunk of code and data that reside in RAM and has at least one thread of control
- Bash is single-threaded and does things sequentially (think turn-based games like chess)

©DENNIS SIMPSON 2013-2022

Shell Process

- Processes are started by other processes when they fork
- Processes inherit their environment from the process that started them - the new process is called the child process and the original process is called the parent process
- Programs often look for environment (sometimes called startup) files to modify their inherited operating environment and provide configuration of the program

©DENNIS SIMPSON 2013-2022

Environment Variables

- Variables are only usable by the process that creates them
- Processes have a way to store variables in their environment, which is a special part of process memory
- Environment variables get copied to child processes
- Environment variables are normally named using capital letters and numbers only
- Variables can be put into, or taken out of, the environment

```
export VARNAME
export VARNAME=“Data”
export -n VARNAME
```

©DENNIS SIMPSON 2013-2022

Login Shell

- When you login to the system, the program that allowed you to login runs whatever shell program is configured for your login account, typically bash for Linux users
- The first shell process started when you log onto a Linux system is called your login shell
- It is used to start other programs, manage files, and observe and control the system
- If you start additional shells such as in terminal windows or by running scripts, they are called interactive non-login shells
- When your login shell process ends, you are logged out

©DENNIS SIMPSON 2013-2022

bash Environment Files

- Environment files are shell scripts that run commands to configure programs, set variables, manipulate files, and/or send messages
- Each user's environment files are kept in their home directory, and users can create, modify, or remove them
- bash runs /etc/profile before running user-specific environment script files, in order to provide a global minimum configuration for all bash users
- bash looks for ~/.bash_profile, ~/.bash_login, ~/.profile and ~/.bash_logout for login shells
- bash looks for ~/.bashrc for interactive non-login shells

©DENNIS SIMPSON 2013-2022

Working With Data in bash

- Working with strings, integers, and variables
- Second challenge script

©DENNIS SIMPSON 2013-2022