Unix Scripting

Week 12

Agenda

- **Variable Expansion**
- **Shell Expansion**

${} vs $()

- ${expression}
  - Is the format for parameter expansion
  - ${parameter#word}
    - Remove Smallest Prefix Pattern. The word shall be expanded to produce a pattern. The parameter expansion shall then result in parameter, with the smallest portion of the prefix matched by the pattern deleted.
  - ${parameter##word}
    - Remove Largest Prefix Pattern. The word shall be expanded to produce a pattern. The parameter expansion shall then result in parameter, with the largest portion of the prefix matched by the pattern deleted.

Example

- Look at the output of the following commands:
  - `x="HELLO"`
  - `echo ${x}`
  - `echo ${#x}`
  - `echo ${x#L*}`

Variable Expansion

- **$var** - "$" before a variable name will retrieve it's value, also called variable substitution
- **${var}** - brace brackets can be used to delineate the variable name from following characters
- **${!var}** - will use value of "var" as the variable whose value will be retrieved

What is the difference between `$name` and `${name}`

- Try this:
  - `name=seneca`
  - `echo $name`
  - `echo $name.txt`
  - `echo $name_txt`
  - `echo ${name}_txt`
  - `echo "${name}"_txt`
- Try this:
  - `name="seneca  college"`
  - `echo $name`
  - `printf "%q\n" $name`
  - `printf "%q\n" "$name"`
- What have you observed?

Variable Expansion

- **${var:-default}** - expands to "default" value if "var" is null or unset, "var" is unchanged
- **${var:+default}** - expands to "default" value if "var" is set, "var" is unchanged
- **${var:?errmsg}** - sends "errmsg" to stderr if "var" is null or unset, "var" is unchanged, if the shell is non-interactive it will exit
- **${var:=default}** - sets "var" to "default" value if "var" is null or unset

Variable Expansion

- **${var:offset:length}** - substring substitution, with an optional length
- positive offset (from beginning of string) for first character is 0
- negative offset (from end of string) for last character is -1
- positive length means length of substring
- negative length is offset from end of string

Variable Expansion

- **${#var}** - gives the length of "$var"
- **${!var*}** - displays all variable names beginning with "var" as a single string
- **${!var@}** - displays all variable names beginning with "var" as separate strings

Variable Expansion

- **${var%pattern}** - removes the shortest part of "$var" that matches "pattern", from the end
- **${var%%pattern}** - removes the longest part of "$var" that matches "pattern", from the end
- **${var#pattern}** - removes the shortest part of "$var" that matches "pattern", from the beginning
- **${var##pattern}** - removes the longest part of "$var" that matches "pattern", from the beginning
  - if variable is "*" or "@", the action is applied to each positional parameter

Variable Expansion

- **${var/pattern/string}** - replaces the longest part of "$var" that matches "pattern", with "string“ defaults to replacing the first match
  - if pattern begins with /, all matches will be replaced
  - if pattern begins with #, the match must be at the beginning of "$var"
  - if pattern begins with %, the match must be at the end of "$var"

Variable Expansion

- **${var^pattern}** - converts the first character of substrings that match "pattern" to uppercase
- **${var^^pattern}** - converts all characters of substrings that match "pattern" to uppercase
- **${var,pattern}** - converts the first character of substrings that match "pattern" to lowercase
- **${var,,pattern}** - converts all characters of substrings that match "pattern" to lowercase
  - if pattern is missing, default is "?", matching all characters in "$var"
  - if variable is "*" or "@", the action is applied to each positional parameter

Shell Expansion

- The shell performs eight kinds of expansion, in the following order: brace expansion
  - tilde expansion
  - variable expansion, command substitution, arithmetic expansion
  - process substitution
  - word splitting
  - pathname expansion

Brace Expansion

- can generate sequences of character strings
- similar idea to pathname expansion, without the need to match existing filenames
- can use a comma-separated list, or a sequence
- will not be expanded within quotes

Tilde Expansion

- can be used as shortcut to some directory names
- ~ is home directory of current user
- ~username is home directory of specified user
- ~+ is the current directory, same as $PWD
- ~- is the previous directory, same as $OLDPWD
- will not be expanded within quotes
- here are some examples of tilde expa

Command Substitution

- **$(command-line)** - new style (korn, bash) of command substitution, easily nested to multiple levels
- **`command-line`** - old style (bourne) of command substitution, nesting requires escaping inner back-quotes, only one level of nesting possible

Arithmetic Expansion

- **$((expression))** - new style (korn, bash) of arithmetic expansion, easily nested

Practice parameter and variable expansion

- Try the following commands:
  - `Fname=(joe bob mary sue)`
  - `declare -p Fname`
  - `printf "%q\n" "${Fname[*]}"`
  - `printf "%q\n" "${Fname[@]}"`
  - `printf "%q\n" "${!Fname[@]}"`
  - `printf "%q\n" "${!Fname[*]}"`

- What have you observed?

Activity:Practice parameter and variable expansion

- Try the following commands:
  - `printf "%q\n" hello "${FOO?Enter Value}"`
  - `FOO=`
  - `printf "%q\n" hello "${FOO?Enter Value}"`
  - `printf "%q\n" hello "${FOO:?Enter Value}"`
  - `printf "%q\n" hello "${COLOR:-blue}"`
- What have you observed?

Shell Expansions

- Expansion is performed on the command line after it has been split into *tokens*.
- There are seven kinds of expansion performed:

• Brace Expansion	Expansion of expressions within braces.
• Tilde Expansion	Expansion of the ~ character.
• Shell Parameter Expansion	How Bash expands variables to their values.
• Command Substitution	Using the output of a command as an argument.
• Arithmetic Expansion	How to use arithmetic in shell expansions.
• Process Substitution	A way to write and read to and from a command.
• Word Splitting	How the results of expansion are split into separate arguments.
• Filename Expansion	A shorthand for specifying filenames matching patterns.
• Quote Removal	How and when quote characters are removed from words.

Activity

- What is the output of the following commands?
  - FILE="example.tar.gz"
  - echo "${FILE%%.*}"
  - echo "${FILE%.*}"
  - echo "${FILE#*.}"
  - echo "${FILE##*.}“
  - basename $FILE .gz

Word Splitting: IFS

- IFS stands : **I**nternal **F**ield **S**eparator
- Any character that appears in $IFS is considered a word separator
- If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset,
  - any sequence of <space>s, <tab>s, or <newline>s at the beginning or end of the input shall be ignored
  - and any sequence of those characters within the input shall delimit a field.

Activity

- Run the followings commands and see what happen:
  - `IFS=":"`
  - `Foo="1:2:3::4"`
  - `echo $Foo`
- What is the output?
- Change IFS=" " and run echo $FOO. What happen?

Activity: Run the following commands and explain each line + how it works

```
echo "enter name, stdID, semester"
read data
#enter Sara, 12345, S2022
IFS=','
read -a stdInfo <<< "$data"
echo "Name : ${stdInfo[0]}"
echo "ID : ${stdInfo[1]}"
echo "Semester : ${stdInfo[2]}"
```

Activity: Run the following commands and explain each line + how it works

```
oldIFS="$IFS"
IFS=":"
result="$(grep -w ^admin /etc/passwd)"
echo "$result"
set -- $result
echo $1
echo $6
for i in $result; do echo "$i"; done
IFS="$oldIFS"
```

More on set command

- `set` allows you to change the values of shell options and set the positional parameters, or to display the names and values of shell variables.
  - Learn more: https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html

Activity

- Do some research about the possibility of using IFS in awk command.
  - Explain how to use `IFS` in `awk` command. Provide an example!
  - Explain how the following works:
    - `echo 1,2,3,4,5 | awk 'BEGIN { FS = "," } ; { print $2 }'`

Some Useful Commands and Options

- `eval [arg]`
- `seq`
- `nl filename`
- `cat -n filename`
- `cat -b filename`
- `cat -A filename`
- `tac filename`
- `cut --complement`
- `fold and fmt`
- `$RANDOM`

`eval` command

- On a Unix or Linux system, the **`eval`** command is used to run the arguments as a shell command.
  - `eval [arg]...`
  - The command evaluates the argument first, then executes the command it contains.
- Example:
  - `COMM="ls"`
  - `eval $COMM`

seq command

- **seq** - generates sequences of numbers:
  - `seq 5`
  - `seq 5  8`
  - `seq 1.3 1.5 10`
- What is the output of the following command?
  - `echo seq{1..10..2}`