Using the shell

Johan Öhlin

3 November 2021

Yabs

grep

```
$ whatis grep
grep (1) - print lines that match patterns
$ cat file.txt
word
hello
swordfish
$ grep -n word file.txt
1:word
3:swordfish
$ grep -ci WORD file.txt
2
$
```

grep

Find exercises from GitHub:

`https://github.com/Najoj/yabsbashcourse/blob/main/exercises/2/grep`

sed

```
$ whatis sed
sed (1) - stream editor for filtering and
transforming text
$ cat file.txt
word
hello
swordfish
$ sed "s/ord/in/" file.txt
win
hello
swinfish
$
```

sed

```
$ sed "2,4d" file.txt
$ sed "s/this/that/" file.txt
$ sed "s/this/that/g" file.txt
$ sed '10,20s/this/that/' file.txt
$ sed '10,20d' file.txt
$ sed 'p' file.txt
$ sed -e '2p' -e '3d" file.txt
$ sed -i "s/this/that/g" file.txt
$ sed -f script-file.sed file.txt
```

sed

Find exercises from GitHub:

`https://github.com/Najoj/yabsbashcourse/blob/main/exercises/2/sed`

awk

```
$ whatis awk
awk (1) - pattern scanning and text processing
language
$ awk "{ print $1 }"
$ awk 'BEGIN { printf "wc -l\n" }
    { s+=1 }
  END { printf "%d\n", s }' file.txt
$ awk "/match/ {...} < file.txt
$ awk -f script-file.awk file1.txt file2.txt
```

awk

Built-in variables and functions	
ARGC	Number of arguments.
NR	Lines read so far.
ENVIRON	Environment variables.
FILENAME	Name of file provided.
getline	Set $0 to next input line.
next	Skip to next line.
nextfile	Skip the rest of this file.

See `man` page for more.

```
$ sed 'END { print NR }' < file.txt
```

awk

Find exercises from GitHub:
`https://github.com/Najoj/yabsbashcourse/blob/main/exercises/2/awk`

find

Does *not* take files as arguments.

```
$ whatis find
find (1) - search for files in a directory hierarchy
$ find /home/user -name "file.txt"
./file.txt
$ find /home -name "file.txt" -exec grep "hi" {} +
$ find / -name "file.txt" -exec grep "hi" {} \
;
$
```

find

Useful options:

`-name n`	Name of file, case sensitive
`-iname n`	Name of file, case insensitive
`-path n`	Path of file, case sensitive
`-ipath n`	Path of file, case insensitive
`-type n`	Type of file. `d` directory, `f` file, etc.
`-maxdepth n`	Descend at most `n` directories
`-mindepth n`	Descend at least `n` directories
`-exec cmd {} +`	Execute `cmd` on each file found

Can use some logic, for example:

```
find /some/dir -name n -or -name m
```

find

Find exercises from GitHub:

`https://github.com/Najoj/yabsbashcourse/blob/main/exercises/2/find`

xargs

```
$ whatis xargs
xargs (1) - build and execute command lines from
standard input
$ cat file.txt
a
b
fileB.txt
$ cat file.txt | xargs ls
a:
fileA.txt

b:
fileB.txt
$ ls a b
```

xargs

Find exercises from GitHub:

`https://github.com/Najoj/yabsbashcourse/blob/main/exercises/2/xargs`

Summary

Use the right tool for the right task.

`sed` and `awk` are languages themselves, we have only touched upon them.

`grep`, `sed`, and `awk` can usually execute the same task.

`find` is a multi-tool; it can do a lot at once.

`xargs` does one thing well.

Summary

Use the right tool for the right task.

`sed` and `awk` are languages themselves, we have only touched upon them.

`grep`, `sed`, and `awk` can usually execute the same task.

`find` is a multi-tool; it can do a lot at once.

`xargs` does one thing well.