Tutorial 2 – Bash & Scripting

18 SEPTEMBER 2020

MICB405

Utilities Review

```
cd (change directory)
ls (-l) (list files) (-long)
pwd (print working directory)
cp (copy)
rm (-r) (remove) (-recursive)
mv (move)
cat (concatenate)
echo (print content you provide as argument)
```

More Utilities…

grep (global regular expression print)

less (less = more

head/tail (self-explanatory)

wc (-l, -w, -m) (wordcount) (-lines, -word, character)

sort (sort lines)

uniq (unique lines)

chmod (change mode)

mkdir (make directory)

Advanced Utility

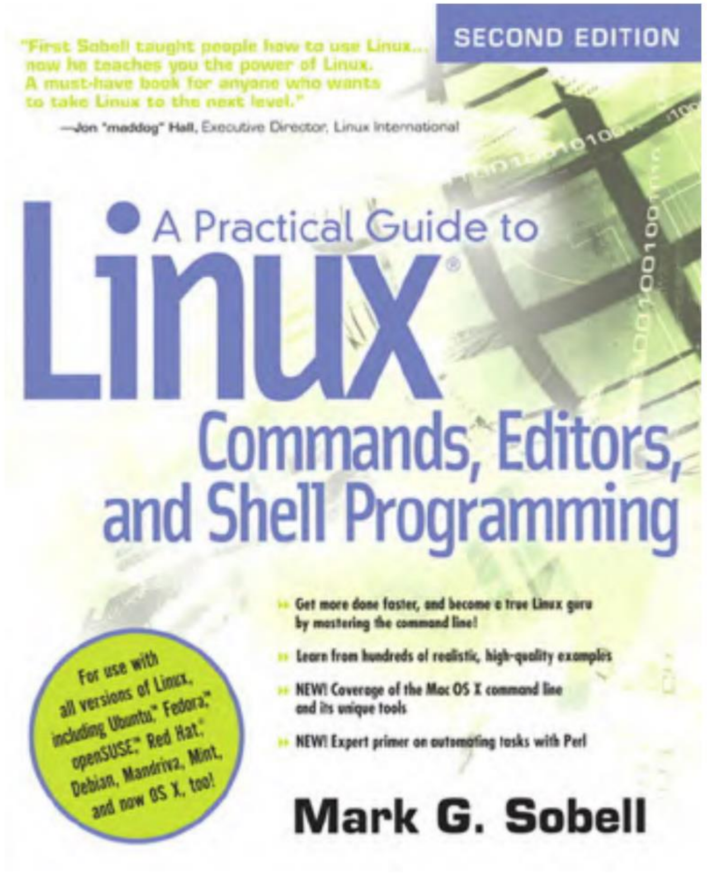
`awk`

Not short for anything – does file processing based on lines & columns

Special Characters

```
> (redirect)
>> (append)
< (redirect, again)
| (pipe) (not 1, l, or I)
; (separate command)
&& (AND) || (OR)
```

Review


SECOND EDITION
"First Sobell taught people how to use Linux... now he teaches you the power of Linux. A must-have book for anyone who wants to take Linux to the next level."
—Jon "maddog" Hall, Executive Director, Linux International
A Practical Guide to Linux®
Commands, Editors, and Shell Programming
Get more done faster, and become a true Linux guru by mastering the command line!
Learn from hundreds of realistic, high-quality examples
NEW! Coverage of the Mac OS X command line and its unique tools
NEW! Expert primer on automating tasks with Perl
For use with all versions of Linux, including Ubuntu,® Fedora,® openSUSE,® Red Hat,® Debian, Mandriva, Mint, and now OS X, too!
Mark G. Sobell

In-terminal Text Editors

`nano` = easiest

- **`nano filename.txt`**

`vim` = more advanced

- Rich keyboard shortcuts but steeper learning curve

These are essential for creating script text files

Scripting Overview

Scripts are files containing a series of directions to help automate routine or repetitive tasks

Scripts at the command line take the form of a text file somewhere in the filesystem that can be "executed"

Headers

All script files need a header to indicate how the script file should be run

First line (“shebang”):

```
#!/bin/bash
```

- Program should interpret following text using Bash
- Other programs = other headers

Variables

We can create a variable and assign it a value with **`results_dir=“results/”`**

- Note that spaces matter when setting Bash variables. Do not use spaces around the equal sign

Variables

To access a variable’s value, we use a dollar sign in front of the variable’s name.

Suppose we want to create a directory for a sample’s alignment data, called <sample>_aln/, where <sample> is replaced by the sample’s name.

```
sample=“Individual_2A”
mkdir “${sample}_data/”
```

This will create a file with the name “Individual_2A_data”

Commandline Arguments

Commandline arguments are assigned to the value $1, $2, $3, etc…

Variables created in your Bash script will only be available for the duration of the Bash process running that script.

Demonstration

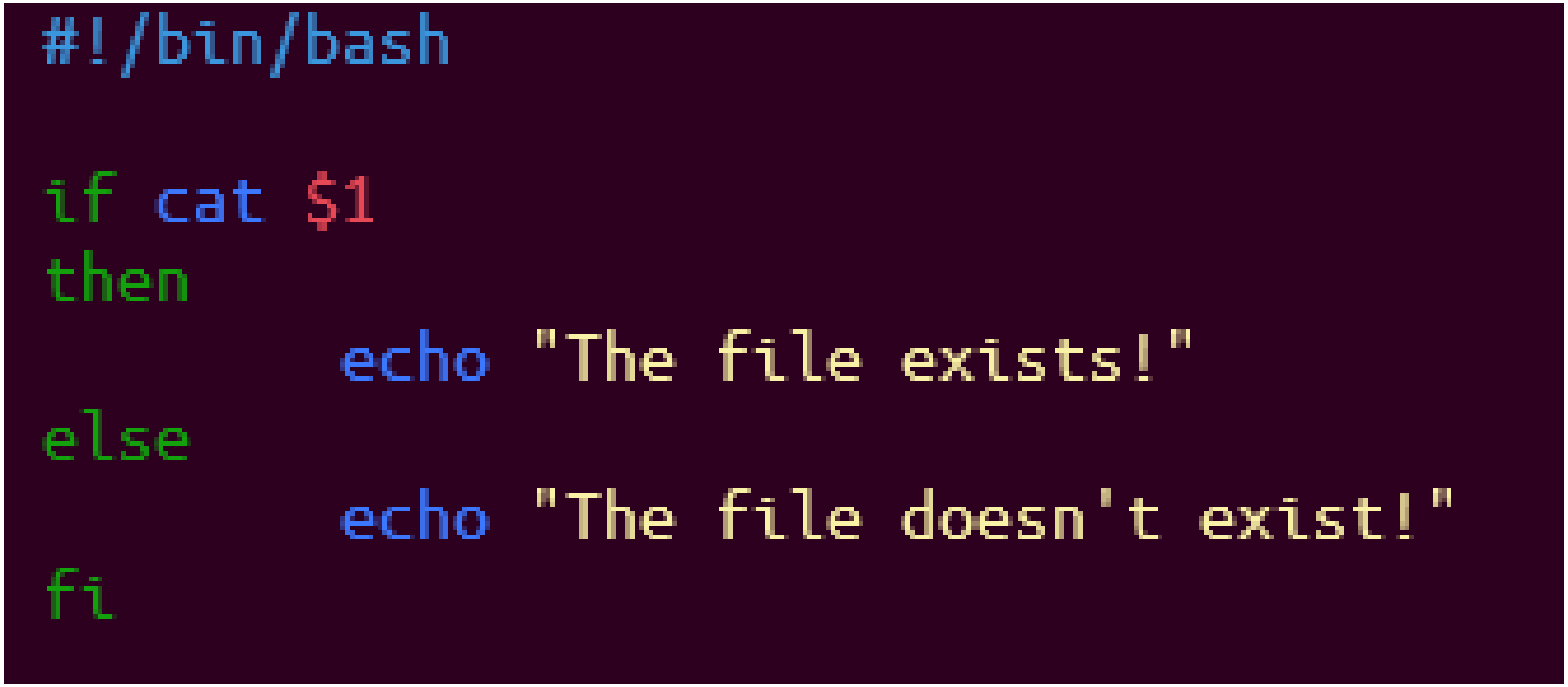
`if` Statements

Bash supports the standard if conditional statement.

The basic syntax is:

```
if [commands]
then
     COMMANDS
else
     DIFFERENT COMMANDS
fi
```

`if` Statement Example



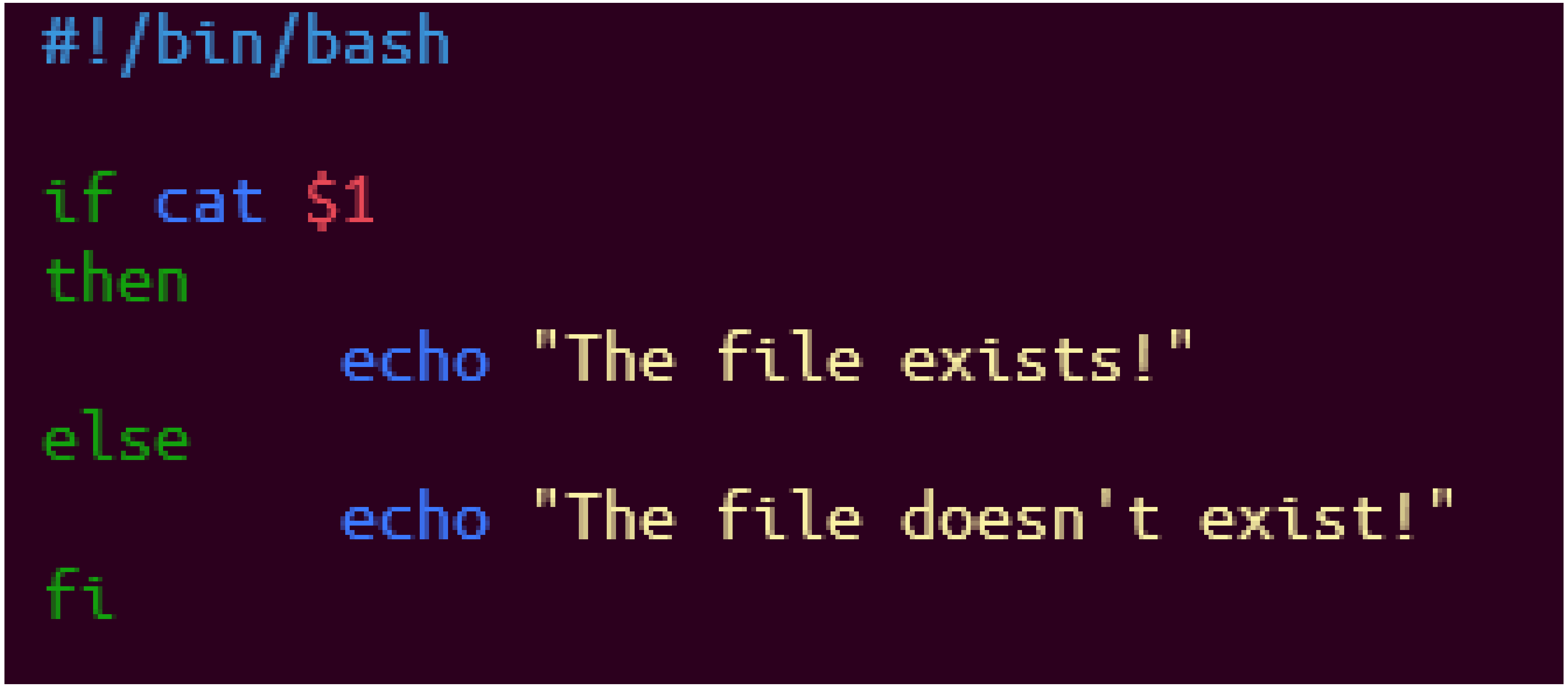
```
#!/bin/bash

if cat $1
then
        echo "The file exists!"
else
        echo "The file doesn't exist!"
fi
```

Return codes

`0` = Command executed successfully

`1 or more` = Command did not execute successfully

`if` Statement Example



```
#!/bin/bash

if cat $1
then
        echo "The file exists!"
else
        echo "The file doesn't exist!"
fi
```

`test` statements

Like other programs, **`test`** exits with either 0 or 1.

Test statements can be included at the beginning of the if program using

However **`test`**'s exit status indicates the return value of the test specified through its arguments, rather than exit success or error.

`test` supports numerous standard comparison operators.

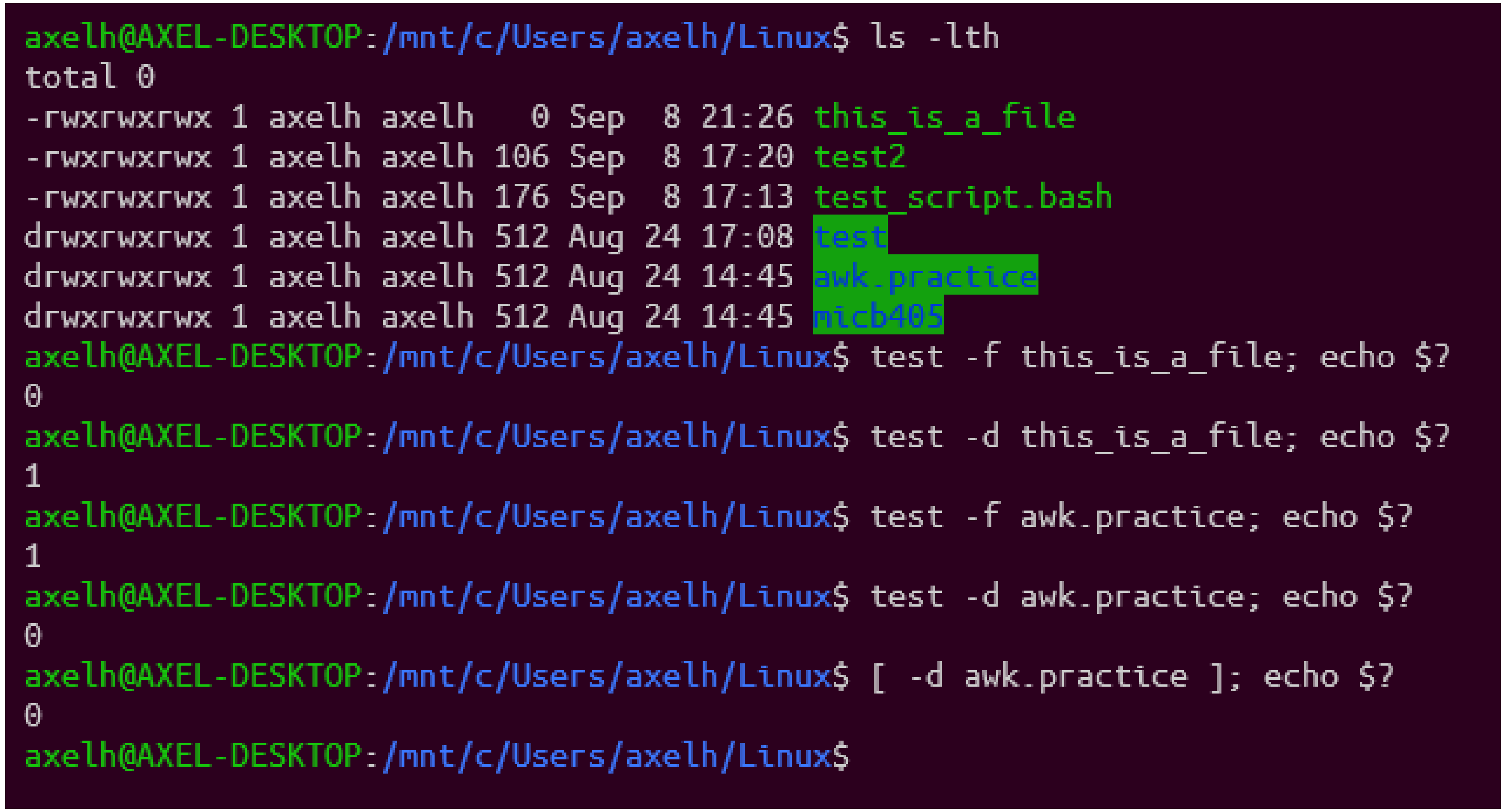
`test` Summary

String/integer	Description
`-z str`	String str is null
`str1 = str2`	str1 and str2 are identical
`str1 != str2`	str1 and str2 are different
`int1 -eq -int2`	Integers int1 and int2 are equal
`int1 -ne -int2`	int1 and int2 are not equal
`int1 -lt -int2`	int1 is less than int2
`int1 -gt -int2`	int1 is greater than int2
`int1 -le -int2`	int1 is less than or equal to int2
`int1 -ge -int2`	int1 is greater than or equal to int2

`test` Summary for Files/Directories

File/directory expression	Description
-d dir	dir is a directory
-f file	file is a file
-e file	file exists
-r file	file is readable
-w file	file is writable
-x file	file is executable

test Example



```
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$ ls -lth
total 0
-rwxrwxrwx 1 axelh axelh   0 Sep  8 21:26 this_is_a_file
-rwxrwxrwx 1 axelh axelh 106 Sep  8 17:20 test2
-rwxrwxrwx 1 axelh axelh 176 Sep  8 17:13 test_script.bash
drwxrwxrwx 1 axelh axelh 512 Aug 24 17:08 test
drwxrwxrwx 1 axelh axelh 512 Aug 24 14:45 awk.practice
drwxrwxrwx 1 axelh axelh 512 Aug 24 14:45 micb405
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$ test -f this_is_a_file; echo $?
0
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$ test -d this_is_a_file; echo $?
1
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$ test -f awk.practice; echo $?
1
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$ test -d awk.practice; echo $?
0
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$ [ -d awk.practice ]; echo $?
0
axelh@AXEL-DESKTOP:/mnt/c/Users/axelh/Linux$
```

Putting it all together…

`if` Statements + **`test`** Statements

Combining test with if statements is simple:

```
#!/bin/bash
if [ -f some_file.txt ]
then
     ACTION TO PERFORM
else
     ACTION TO PERFORM
fi
```

- Note the spaces around and within the brackets: these are required.

`for` loops

In bioinformatics, most of our data is split across multiple files.

Many processing pipelines need a way to apply the same workflow on each file, taking care to keep track of sample names.

Looping over files with Bash’s for loop = simplest way to accomplish this

Three essential parts to creating a pipeline to process a set of files:

1. Selecting which files to apply the commands to
2. Looping over the data and applying the commands
3. Keeping track of the names of any output files created

`for` loops

```
#!/bin/bash
for (variable) in (directory/array/etc.)
do
    ACTION TO PERFORM
    ANOTHER ACTION TO PERFORM
done
```

for looping through directories

Creating **for** loops that loop through an entire directory

```
Summarize_files.bash
#!/bin/bash
for foo in /home/axelh/Documents/data/*
do
    head ${foo} > ${foo}.head.and.tail.txt
    tail ${foo} >> ${foo}.head.and.tail.txt
done
```

for looping through directories

```
#!/bin/bash
for foo in /home/axelh/Documents/data/*
do
      if [ -f ${foo} ]
      then
            head ${foo} > ${foo}.head.and.tail.txt
            tail ${foo} >> ${foo}.head.and.tail.txt
      else
            echo “${foo} is not a file to summarize!”
      fi
done
```

Breakout Groups