Introduction of Linux

赵晓臻

> oslab2019_class1@163.com

刘宇芳

> oslab2019_class2@163.com

PART II

- Shell Script
- Compile & Debug (for C)
- Text Editor (Vim, Sublime text, Atom)

Shell Script

A shell script is a program designed to be run by the shell. The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, etc. is called a warpper .

Variable

Define, Assignment & Read

```
VariableName=value
read VariableName
```

- no space between VarName and the equality sign
- first letter: a-z A-Z
- no keywords of shell

Use a variable

```
$VariableName
${VariableName}
```

Special Variables

```
$0  # filename of the script
$n  # the n-th argument
$#  # the number of the arguments
$HOME  # user directory
$$  # PID
```

Examples:

test1.sh

```
#!/bin/bash
read a
read b
c=$[($a+$b)**$a]
echo $c
```

with arguments

```
#!/bin/bash
echo $[($1+$2)**$1]
```

String

single quotes

```
str='no variables'
```

double quotes

```
v='variables'
str="$v or \"escape character\""
```

connecting

```
str1="connecting strings"
str2="simple"
str3=$str1" is "$str2
```

string length

```
${#string}
```

substring

```
${string:begin:end}
```

Example:

```
#!bin/bash
str="alibaba is a great company"
echo ${#str}
echo ${str:1:4}
```

printf

differences from “printf” in C

- no ( )
- using space between two arguments

if the number of arguments is greater than the number of % in format, the format-string will be reused repeatedly

```
printf “%s %s\n” 1 2 3 4
```

output:

```
1 2
3 4
```

Branches

```
if [condition]
 then
      …
      else
           …
fi
```

or

```
if [condition1]; then
      …
 elif [condition2]; then
      …
      else
      …
fi
```

Operator

Numerical Comparison Operators

Operator	Remark
-eq	==
-ne	!=
-gt	>
-lt	<
-ge	>=
-le	<=

Other Operators

Operator	Remark
=	== for string
!=	!= for string
-z	If the string is empty
-f / -d	is file / is dir.
-r / -w / -x	check permission
-e	if a file/dir. exists

Example:

```
#!/bin/bash
YACCESS=`date -d yesterday +%Y%m%d`
FILE="access_$YACCESS.log.tgz"
if [ -f "$FILE" ];then
      echo "OK"
 else
    echo "error $FILE"
fi
```

Loop

```
for variable in list
 do
      …
done
```

```
while [ condition ]
 do
      …
done
```

```
break
continue
```

Example:

```
for FILE in $HOME/*
 do
      echo $FILE
done

count=0
while [ $count –lt 5 ]
 do
   count=$[$count+1]
   echo $count
done
```

PART II

- Shell Script
- Compile & Debug (for C)
- Text Editor (Vim, Sublime text, Atom)

Compilation & Execution

`GCC` (GNU C Compiler → GNU Compiler Collection)

```
gcc test.c    # compile the C source file
```

produce an executable file named (by default) `a.out`

```
./a.out       # run the program a.out
```

Useful Options

```
gcc -o test test.c
gcc -g -o test test.c
gcc test.c -g -o test
```

Separate Compilation

complie a program with several separate files

```
gcc -c test1.c
gcc -c test2.c
gcc test1.o test2.o -o test
```

`-c` : compile to produce an object file, which is not executables just machine-level representations of the source code.

Linking with Libraries

Library

`lib+name.a`	(-static)
`lib+name.so`	(default)
`-l+name`	Link with libraries manually
`-L+lib’ s dir`	Give the directory manually

```
gcc hello.c -shared -o libhello.so
gcc test.c -lhello -L. -o test
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

```
gcc hello.c -c -o hello.o
ar -r libhello.a hello.o
gcc test.c -lhello -L. -static -o test
```

make↔Makefile

Build the program automatelly according to the makefile.

Makefiles are based on rules as:

```
target [target ...]: [component ...]
Tab⇄ [command 1]
.
.
.
Tab⇄ [command n]
```

```
hello.o: hello.c hello.h
Tab⇄ gcc hello.c -c -g
```

Debugging with GDB (GNU debugger)

`gdb` **Enter the gdb environment.(gcc test.c -g -o test)**

Command	Remark
file [file name]	load a excutable file
r	run
c	continue
b [line number] b [function name]	set Breakpoint
s, n	excute a line of source code
p [variable name]	print the value of a variable
q	quit
help [command]	

PART II

- Shell Script
- Compile & Debug (for C)
- Text Editor (Vim, Sublime text, Atom)

Recommanded Editors

Sublime

Atom

Vim(CLI)

Superorities

Cross-platform

Extensible

Lightweight

Sublime

A sophisticated text editor for code, markup and prose

source: http://www.sublimetext.com/

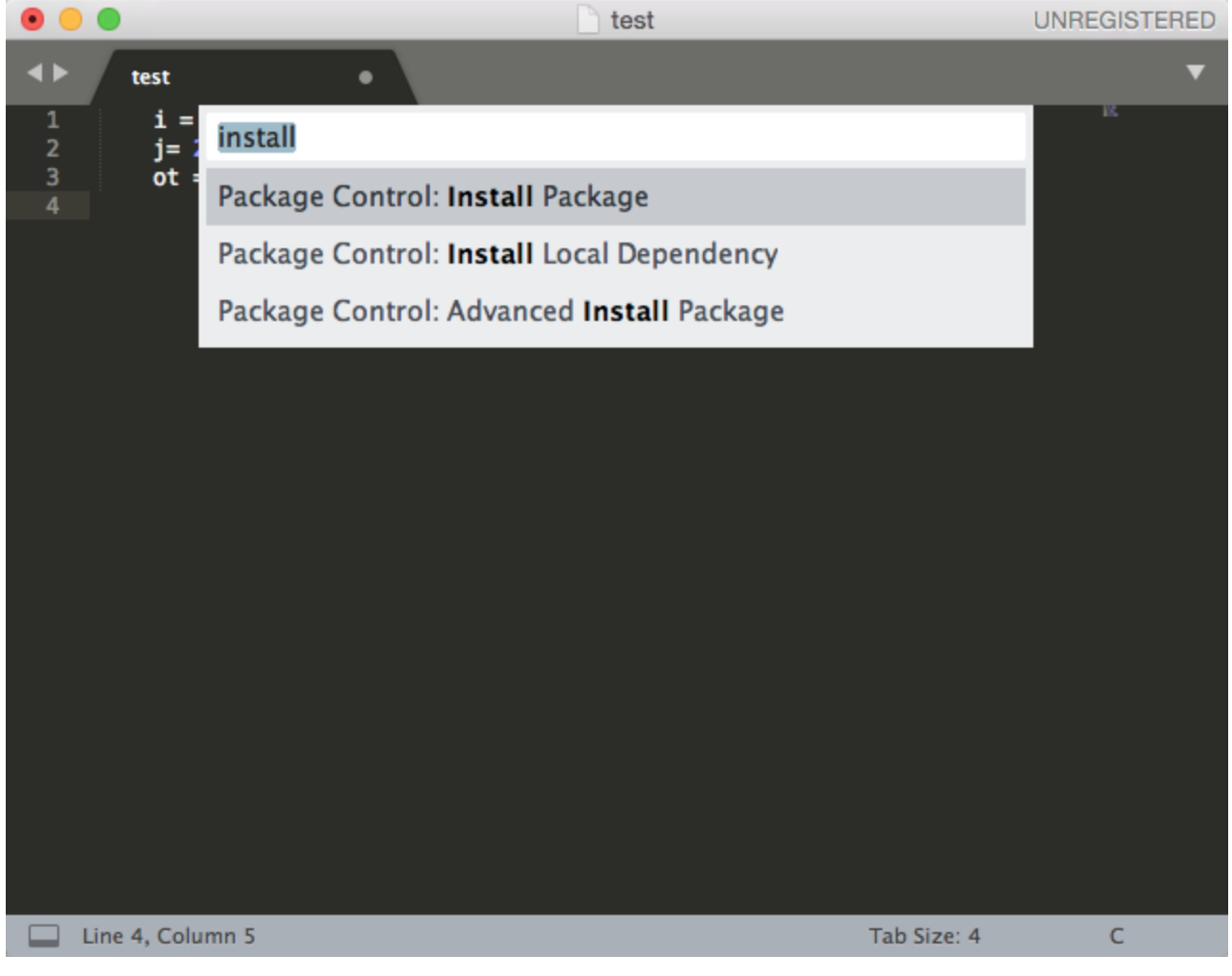
Installation for Linux

via Package Manager(apt-get)

```
sudo add-apt-repository ppa:webupd8team/sublime-text-3
sudo apt-get update
sudo apt-get install sublime-text-installer
```

Package Control

- go to Command Palette (ctrl+shift+p)
- type *install*
- you will see a list of plugins



Plugins

To see the list of plugins(Preferences=>Package Settings)

Alignment

For code alignment(ctrl+alt+a)

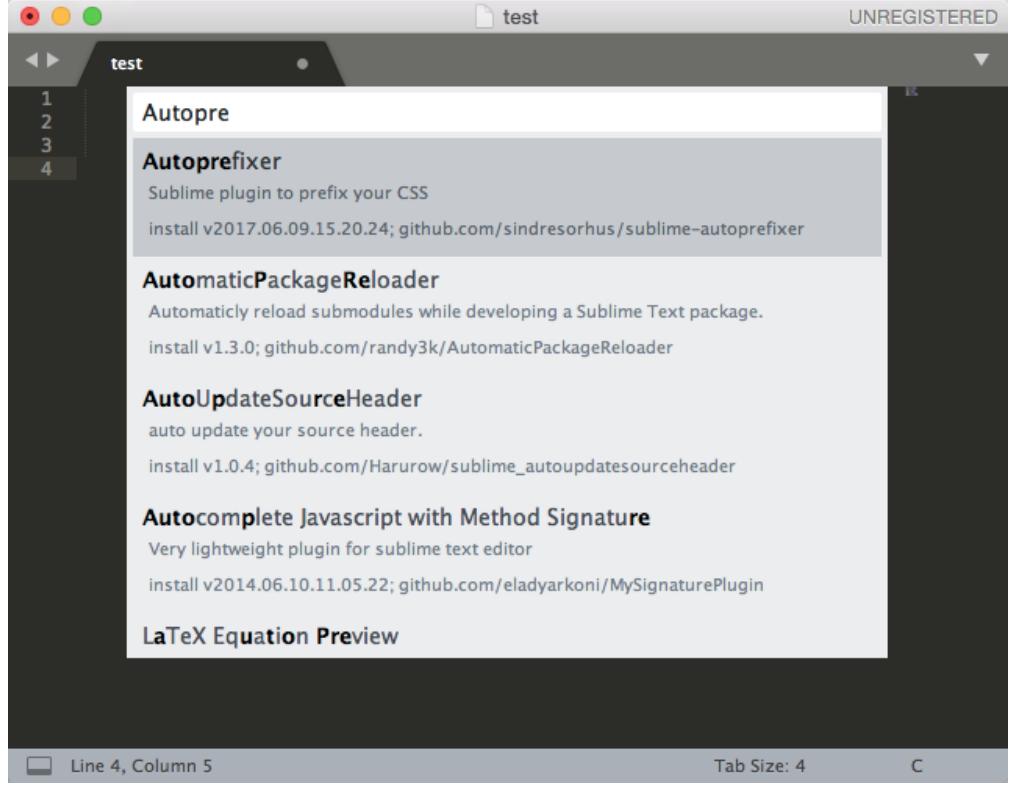
BracketHighlighter

For code highlighting

DictionaryAutoComplete

For dictionary completing

...



Atom

A hackable text editor for the 21st Century

source: https://atom.io/

Similar to Sublime

Installation for Linux

via Package Manager(apt-get)

```
sudo add-apt-repository ppa:webupd8team/atom
sudo apt-get update
sudo apt-get install atom
```

Vim 

Vim is a highly configurable text editor built to make creating and changing any kind of text very efficient.

Installation for Linux

via Package Manager(apt-get)

```
sudo apt-get install vim
vimtutor # obtain a vimÃ¢â‚¬â„¢s tutorial
```

Creat a file

```
vim filename
```

Three Modes

Command Mode

all the keys are bound to commands (typing "j" -- it will move the cursor down one line)

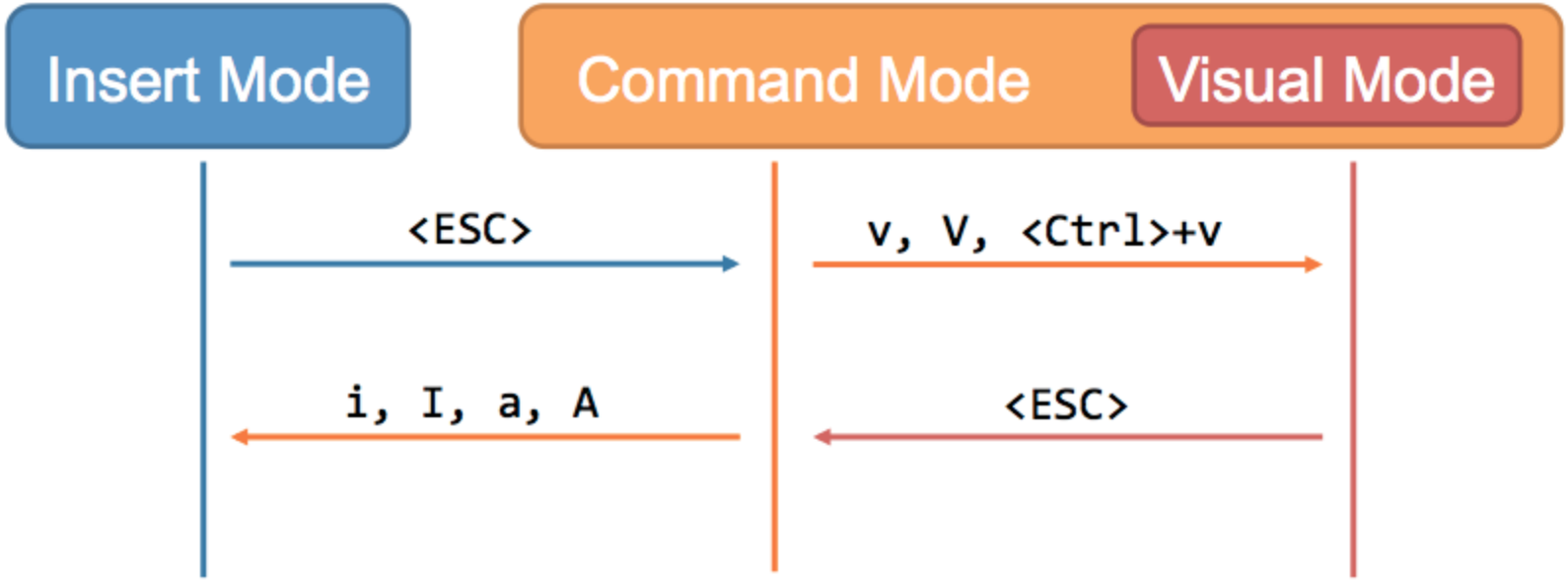
Insert Mode

all the keys are exactly keys (typing "j" -- inserting "j")

Visual Mode

helps to visually select some text, may be seen as a submode of the the command mode

Three Modes



Keys in command mode

Quit and Save

- `w` write the current buffer to disk (save)
- `q` close the current window
- `x` or `wq` save and close
- `q!` close without save

Scroll the Screen

downwards

- *ctrl + f* 1 page
- *ctrl + d* 1/2 page
- *ctrl + e* 1 line

upwards

- *ctrl + y* 1 page
- *ctrl + u* 1/2 page
- *ctrl + b* 1 line

Movement of the Cursor

- `h` moves the cursor one character to the left.
- `j` moves the cursor down one line.
- `k` moves the cursor up one line.
- `l` moves the cursor one character to the right.
- `0` moves the cursor to the beginning of the line.
- `$` moves the cursor to the end of the line.
- `w` moves forward one word.
- `b` moves backward one word.
- `G` moves to the end of the file.
- `gg` moves to the beginning of the file.
- `` `. `` moves to the last edit.