CS 35L Fall 17 Section 7 Notes 2
Zhaowei Tan

More command lines!

1. > <: Redirect the output to file/redirect file as input

2. cmd1| cmd2: Pipeline takes the output of cmd1 as input of cmd2

3. find and grep: Help you search in system/file

4. Wildcard: ? and *

Bash basics:

1. To declare a file as a bash script, starting with #!/bin/bash, so that when you execute it, the system knows it is a bash script, and run it accordingly.

2. Comment in bash: #

3. To assign a variable: var=5, no need to declare it beforehand. To print it, echo $var. Pay attention to when you need the dollar sign.

4. For arithmetic calculation, use $(()). Example: i=$(($i + 1))

5. Inside shell, you can wrap your command program inside `. For example, a=`ls /usr/bin`. Alternatively, you could use a=$(ls /usr/bin)

6. String could be wrapped inside " or ', there's a major difference though. if we have lan=English
   echo "Language is $lan" and echo 'language is $lan'
   have different outputs.

7. Some built-in variables:
   $0: the first command line argument (i.e. your program name)
   $1: the second command line argument
   $2: similar
   …
   $#: the number of arguments

8. Get input from user: read variable_name
   Give user prompt: -p
   Example: read –p "your prompt" variable_name

9. If statement: (you need the space)

```
if [ condition ]
then
 commands
elif [ condition ]
then
 commands
else
 commands
fi
```

10. Some useful conditions:

```
string1 == string2 # if two strings are identical
integer_a –eq integer_b # if a equals to b
```

Similarly, we have –gt, -ge, -lt, -le for greater than, greater than or equals to, less than, less than or equals to, respectively

11. While loops:

```
while [ condition ]
do
 commands
done
```

12. For loops:

```
for i in list
do
 commands
done
```

Here the list could be a string with items separated with space
So this makes sense: for i in `ls .`