**Lecture Notes**

Note: Recording w1, w2, w3, w4 done!

**Filter**

* ? = matches exactly a single character
* wc = word count
  + wc -l = # of lines
  + wc -w = # of words
  + wc -c = # of characters
* instructions with no file after the command, scan from the input
* \* = repetition
* | = alternation/pipeline
* \ = remove the special meaning of a character
* (repetitThis) = used for grouping
* . = matches any single character
* [listOfCharacters] = matches any single character from the list
* [first-last] = for a range of characters
* [^listOfCharacters] = the matching can be inverted
* Other characters lose their special meaning inside bracket expression
* egrep -i = ignore case
* egrep -v = invert match
* egrep -w = matches whole word
* egrep -x = matches whole line
* ^ = the start of the line
* $ = the end of the line
* p\* = zero or more repetitions of p
* p+ = one or more repetitions of p
* p? = zero or one occurrence of p
* cut -c2-5, 12, 23- = get the characters of these positions
* cut -d\| -f2 = get the second field of characters separated by |
* head/tail = print the first or the last 10 lines
  + head -1 = print the first line
* sed -e = add the script to the commands to be execute
  + e.g. sed -e ‘EditCommands’ DataFile
* sed -f = add the contents of script-file to the commands to be execute
  + e.g. sed -f EditCommandFile DataFile
* sed ‘s/^/z/;s/$/@ad.unsw.edu.au/’ = add z at the beginning and append @ad.unsw.edu.au at the end

**Shell**

* echo $PS1 = shell command prompt are stored in the variable PS1
* echo $PS2 = secondary command prompt, normally >, is kept in the variable PS2
* echo $HOME = HOME is where you are placed whenever you log on to the system
* echo $PATH = Whenever you type in the name of a program to be executed, the shell searches a list of directories until it finds the required program. This list of directories is contained in a special variable called PATH.
* #!/bin/sh = ‘#!’ says “give the content to the file ‘/bin/sh’”
* #!/usr/bin/python3 = is a python3 program
* chmod 755 filename = prevent user accidently doing things
* chmod u+x, g+x, o+x = chmod 755
* which = locate a command
  + e.g.1
    - which echo
    - > /bin/echo
* $0 = the name of the command
* $\* = all of the command-line arguments (together)
* $@ = all of the command-line arguments (separately)
* $$ = process ID of this shell
* #!/bin/bash = is a particular shell
* rm -rf = force remove directories and their contents recursively
* set -x = shell debugging
* sleep 1 = delay for 1 second
* 1>&2 = send message to standard error
* echo -n = do not output the trailing newlines
* echo -e = enable interpretation of the backslash escapes
* test -r FILE = FILE exists and read permission is granted
* test -w FILE = FILE exists and write permission is granted
* test -x FILE = FILE exists and execute (or search) permission is granted
* diff file1 file2 = compare files line by line

**Perl**

* # !/usr/bin/perl
* ./file.pl = perl file.pl = perl -e <code here>
* ; = is used to separate statement, in C is used to terminate.
* $ = states a scalar variable
* ‘Single quotes’ = in Perl, no variable interpretation.
* ‘print’ in Perl = ‘printf’ in C, printf also can be used in Perl
* <> = read line function
  + <STDIN> = read from standard input, same as <>
* chmop $variable = remove newline
* #!/usr/bin/perl -w = turn on warnings
* the digits in the string will be convert
  + e.g.1
    - print “6”\*”7”
    - >42
* last = same as ‘break’ in C
* If string a empty, is false; if string is 0 and newline, isn’t false; if is just 0, is FALSE (use chmop to remove the newline)
* Variables don’t have to be initialized
* . = strcat
  + e.g.1
    - $a = “Hey”
    - $a = $a.$a
    - print $a
    - > HeyHey
* @a = (1,2,3,4,5) = an array
* $a[] = an array with a index, 0 based indexing
* $#a = the index of the last element
* length $string = same as strlen in C
* substr($string, offset, length) = extracts a substring out of $string and returns it. First character is at offset zero, if offset is negative, starts that far back from the end of the string. If length is omitted, returns everything through the end of the string.
* $a = <STDIN> = single value expected, it reads one line
* @a = <STDIN> = multiple values expected, it reads as many lines as it can, and returns as a list, each line is a element.
* $n = @a = the length of the array -- @a
* put the array into a string = is interpret the element with spaces between the elements
  + e.g.1
    - @a = (0..5);
    - print @, “\n”;
    - >012345
  + e.g.2
    - @a = (0..5);
    - print “@a\n”;
    - >0 1 2 3 4 5
* e.g.1 “cp”
  + open IN, ‘<’, $ARGV[0] or die “$!” # open file passed from argument0, ‘<’ means read it. ‘die’ stop execution. In other words, check for return. ‘$!’ is information mark. Here print some information.
  + open OUT, ‘>’, $ARGV[1] or die # open file passed from argument1, ‘>’ means print out.
  + @a = <IN>
  + print OUT @a
* reverse @array = reverse function, reverse the order of elements
* pop @array = remove the last element from the array
* split /”#”/, $line = split a file with “#” (3 characters here)
* $line =~ s/abc/bbc/g = change ‘abc’ to ‘bbc’
* $line =~ /abc/ = extract ‘abc’
* $line =~ /”(.\*)”.\*([0-9]\*)/ or die =$1 is the first part, $2 is the second part, and check if it succeed
* $hash{$some\_key}
  + foreach $some\_key (sort %hash) = sort
  + foreach $some\_key (keys %hash)
  + foreach $some\_key (sort keys %hash) = ???
* match as much as possible, \*? match as less as possible
* open F, “-|”, “date”; = ‘-|’ get data from a program
  + e.g.1
    - open F, “-|”, “date” or die
    - //Equivalent to
    - if ( ! open F, “-|”, “date”) {
    - die;
    - }
* join “ “, $line = join several string into a single string
  + e.g.1
    - join (“-“. “one”, “two”, “three”); //or join “-“. @arrayOfString;
    - >one-two-three
* split ( /:/, “a:b:c:d:e”, num); = then print out “a b c d e”, is an inverse of join, num means split num of times.
* \D = not a digit, same as /s, /t etc
* -? = optionally ‘-‘
* s/abc/def/ig = ‘g’ flag will change all matchings in the line, otherwise only change the first match in the line

= ‘i’ in the end means ignore the case

* @line vs $line = these two means different things
* print OUT @line = write @line to OUT file
* \w = word
* \S = nonwhite space
* sub func1{
* print “Hello world!\n”;
* }
* &func1(); //& means executable
* &func1();
* >Hello world!
* >Hello world!
* The first argument pass to that function is $\_[0], second is $\_[1], etc
* Or assign them like this:
  + ($message, $n) = @\_
  + //now $message is $\_[0] and $n is $\_[1]
* &function(argv1, argv2, argv3);
* In Perl, all variables are globle
* But ‘my $i’ means variable $i is local to that function
* e.g.1
* sub func{
* my @x = @\_[0..10]
* my @y = @\_[11..20]
* foreach $i (0..$#x){
* …
* }
* }
* print func(100..110, 150..160)
* e.g2
* sub func{
* my ($x, $y) = @\_;
* @x = @$x;
* @y = @$y;
* }
* @a = 1..5
* @b = 10..15
* func(“a”, “b”)
* e.g.3
* $a = 24;
* func{
* my $ref = @\_;
* $$ref = 42; // like a pointer
* //If pass an array
* $ref->[0] = 42
* //@a = 1 .. 10;
* //>a = 42 2 3 4 5 6 7 8 9 10
* }
* func(\$a); // \$a is reference, like pointer in C world
* print “a = $a\n”;
* e.g.4 recurrsive
* return 0 if !@list;
* return shift(@list) + sum\_list(@list);
* eval $list; //evaluate the list which is 1+2+3 and get 6
* Perl return the last value in the subroutine
* Filter\_odd \&even, @a
* //’\’ is pointer
* //& refer to a function called sub even;
* //So inside sub func, &$\_[0] refer to this ‘even’ function
* foreach (@list)
* thus, $\_ is by default
* Perl function -- grep
* @output = grep &even, @a;
* // print the elements from the list that produce request value
* e.g.5
* sub my\_push(\@$){ // except to see an array and a variable
* }
* my\_push @a, 12;
* $aref = [1,2,3,4];
* print @aref; //displays whole array
* … $$aref [0]; //access the first element
* … ${$aref} [1] // access the second element
* … $aref -> [2] // access the third element
* [1,2,3] gives a reference to (1,2,3)
* {a=>1, b=>2} gives a reference to (a=>1, b=>2)
* #!/usr/bin/perl -wi = -i means replace the file, thus no need of $file =~
* or print back to the file
* #!/usr/bin/perl -wip = get rid of while loop to read each line
* @number = sort {$x ⬄ $y} @number;// numeric sorting, ascending order
* @number = sort {$a cmp $b} @number // string comparison, alphabetic order sorting, ascending order
* $number = rand 28 = got a number between 0 and 28

**Python**

* Perl: $s = “string” Python: s = “string” (type is ‘str’)
* Perl: @a = (1,2,3) Python: a = [1,2,3] (type is ‘list’)
* Perl: %a = (1=>’a’, 2=>’b’, 3=>’c’) Python: a = {1:’a’, 2:’b’, 3:’c’} (type is ‘dict’)
* Perl: $a[42] = ‘answer’ Python: a[42] = ‘answer’
* Perl: $h{‘answer’} = 42 Python: h[‘answer’] = 42
* Perl: $x =~ s/a/b/ Python: x = re.sub(‘a’, ‘b’, x)
* Python: print(‘%d %s’ % (count[word], word))
* Def square(x):
  + Return x\*x;
* Call it: print(square(42))
* OR
* Def square(x=42):
  + Return x\*x
* Call it: print(square())
* For … in … :

**Unix Shell Programming 3rd Edition**

**Paste**

* paste file1 file2 = appending file2 after file1 (each line), separated by tab
* paste -d’+’ file1 file2 = separated by +
* paste -d’ ‘ -s file = paste all lines from ‘file’ in a line and separated by a space

**Sed**

* sed ‘s/before/after/g’ file = replace ‘before’ by ‘after’ of ‘file’
* sed ‘s/ .\*$//g’ file = delete the last word of the line
* sed -n ‘1,2p’ file = print the first two lines
* sed -n ‘/UNIX/p’ file = print the lines containing ‘UNIX’
* sed ‘1,10s/unix/UNIX/g’ file = replace ‘unix’ by ‘UNIX’ of the first 10 lines
* sed ‘1,2d’ file = delete the 1 and 2 lines
* sed ‘/UNIX/d’ file = delete from lines containing ‘UNIX’
* sed ‘/jan/s/-1/-5/g’ = replace -1 to -5 of the lines containing ‘jan’

**Tr**

* tr before after < file = replace all ‘before’ to ‘after’
* tr -s ‘ ‘ ‘ ‘ < file = squeeze the multiple spaces to a single space
* tr -d ‘ ’ < file = delete ‘ ‘ (space) of ‘file’

**Sort**

* sort -u file = eliminate duplicate lines from the output
* sort -r file = reverse the order of sort
* sort -n file = sort arithmetically
* sort +5n -t’:’ file = sort ‘file’ skip the first 5 fields separated by ‘:’ and sort numerically

**Uniq**

* uniq file = print unique lines of ‘file’
* uniq -d file = list duplicate lines of ‘file’
* uniq -c file = count line occurrences

**Shell Script**

* chmod +x file(s) = make it executable
* chmod 755 file(s) = make it readable and executable
* # = comments
* variable=value = store a value inside variable (NOTE: no space on either side of the equal sign)
* echo “ $variable” = displaying the value of the variable
  + e.g.1
    - value1=1
    - value2=$value1
    - #Then value2=1
  + e.g.2 value is null
    - echo :$none:
    - #Then print ::
  + e.g.3
    - data=”” or data=’’
    - #Indicate data is null
    - #But
    - data=” “
    - #Indicate data is a space
* mv $filename ${filename}X = rename filename by filenameX
* echo $((a = a+1)) = # a is not set at first and then a becomes 1
* echo a=$((a+1)) = # same as above
* result=$((a > 10 && b < 12)) = # if true result=1, false result=0
* All special characters are ignored by the shell if they appear inside ‘single quotes’.
* Dollar signs are not ignored means that variable name substitution is done by the shell inside double quotes.
  + e.g.1
    - x=\*
    - echo $x
    - > file1 file2 file3 …
    - echo ‘$x’
    - > $x
    - echo “$x”
    - > \*
* Double quotes can be used to hide single quotes from the shell, and vice versa.
  + e.g.1
    - x=” ‘Hello,’ he said”
    - echo $x
    - > ‘Hello.’ he said
    - y=’ “Keeping the logins from lagging,” Bell Labs Record’
    - echo $y
    - “Keeping the logins from lagging,” Bell Labs Record
* Same as single quotes, backslash can remove the special meaning of a character
* Backslash for continuing the lines
  + e.g.1
    - x=one\
    - >two
    - echo “$x”
    - >onetwo
* `command` = execute ‘command’
* $(command) = same as back quotes
* expr = arithmetic operation (+, -, \\*, /, %)
* $# = gets set to # of arguments that were typed on the command line
* $\* = all the arguments passed to the program
* /tmp = is a directory on all Unix systems that anyone can write to. Each time the system gets rebooted, all the files in /tmp are usually removed.
  + e.g.1
    - cat rem
    - > egrep -v “$1” file > /tmp/file
    - > mv /tmp/file file
    - rem ‘xxx’
    - # Then xxx does not in the file
* ${n} = if you supply more than 9 arguments to a program
  + i.e. “$1”, “$2”, ..., ”$9”, “${10}”, “${11}”, …
* shift = left shift your positional parameters
* $? = is set by the shell to the exit status of the last command executed
  + 0 is succeed, others are failure
* /dev/null = system garbage can
* test ‘expression’
  + Operator Returns TRUE (exit status of 0) if
  + string1 = string2 identical
  + string1 != string2 not identical
  + string not null
  + -n string not null (and string must be seen by test)
  + -z string null (and string must be seen by test)
* [expression] = test
* Integer operations
  + int1 -(eq|gt|ge|ne|lt|le) int2
* Test files operators
  + Operator Returns TRUE (exit status of 0) if
  + -d file file is a directory
  + -e file file exits
  + -f file file is an ordinary file
  + -r file file is readable by the process
  + -s file file has nonzero length
  + -w file file is writable by the process
  + -x file file is executable
  + -L file file is a symbolic link
* ! = logical negation operator
* -a = logical ‘and’ operator
* -o = logical ‘or’ operator
* exit = immediately terminate execution of shell program
  + exit 1 = exit failure
* The if else command
  + if command1
  + then
  + command
  + command
  + …
  + elif command2
  + then
  + command
  + command
  + …
  + elif commandn
  + then
  + command
  + command
  + …
  + else
  + command
  + command
  + …
  + fi
* The case command
  + case value in
  + pat1|pat5) command
  + command
  + …
  + command;;
  + [A-Z]|pat2) command
  + command
  + …
  + command;;
  + …
  + patn) command
  + command
  + …
  + command;;
  + #default
  + \*)
  + command;;
  + esac
* sh -x filename command = debugging programs
* : = the null command
* command1 && command2 = if command1 is true, command2 will be executed. Otherwise, command2 gets skipped
* command1 || command2 = if command1 is false, command2 will be executed. Otherwise, command2 gets skipped
* The for command
  + for var in word1 word2 … wordn
  + do
  + command
  + command
  + …
  + done
* $@ = displays all the arguments typed on the command line, one per line
* for var = for var in “$@”
* The while command
  + while commandt
  + do
  + command
  + command
  + …
  + done
* The until command
  + until commandt
  + do
  + command
  + command
  + …
  + done
* break = exit from the loop
* continue = remaining commands in the loop to be skipped
* & = send it to the background for execution
* /dev/tty = refer to the terminal
* echo escape characters
  + \b = backspace
  + \c = the line without a terminating newline
  + \f = form feed
  + \n = newline
  + \r = carriage return
  + \t = tab
  + \\ = backslash character
  + \0nnn = the character whose ASCII value is nnn, where nnn is a one- to three-digit octal number
* read variables = the shell reads a line from standard input and assigns to variables
* printf = same as C programming
* Build-in function
* abs(x)
  + return absolute value
  + integer or floating point number
  + return magnitude if is a complex number
* all(iterable)
  + return True if it’s not empty or 0, otherwise False
* any(iterable)
  + inverse of all(iterable)
* asci(object)
  + return a string containing a printable representation of an object
* bin(x)
  + convert an integer number to a binary string
* class bool([x])
  + return Boolean value (True or False)
* chr(i)
  + for example, chr(97) returns the string ‘a’

always check the length of parameters

$variable = substr $variable, 0, 64;

remove all but necessary characters

$name = s/[^a-z0-9]//g;

#!/bin/sh

if test $REQUEST\_METHOD = “GET”

then

parameters=”$QUERY\_STRING”

else

read parameters

x=`echo $parameters|sed ‘s/.\*x=//;s/[^0-9\-\.\+]//g’`

cat <<eof

Content-type: text/html

<DOCTYPE html>

<html lang=”en”>

<head>

<title>print prime</title>

<body>

eof

….

Cat <<eof

<form method=”GET” action=””>

<p>sum</p>

</form>

</body>

</html>

eof