CPS393 Summary Part 1: Unix/Linux

Week 1

cat f1	display f1 content on stdout
tac f1	display f1 content on stdout, backward
more f1	paginates f1
less f1	paginates, allows forward and backward movement
mkdir, rmdir, cp, mv	make a directory, remove a directory (empty only), copy, move (works for renaming)
wc	size information: word count, line count, etc
chmod	change mode (of a file or dir) chmod +x f1 ## adds execute option for everyone chmod uo+w f1 ## adds write option for user (owner) and other chmod uo=r f1 ## sets user (owner) and other only to read chmod 160 f1 ## sets user to 1 (--x), group to 6 (rw-), other to 0 (---)
stdout and stderr	> f1 changes output to create/overwrite f1 >> f1 changes output to create/append f1 2>/dev/null changes any error outputs to not display at all

Week 2

shell patterns	?, *, [...], etc
regex patterns in shell	must use shopt -s extglob *(exp) 0 or more +(exp) 1 or more ?(exp) 0 or 1 @(exp1|exp2|...) 1 or 2 or .... !(exp) anything that doesn't match exp
regular expressions	. any char * 0 or more of previous char ^ beginning of line $ end of line [...] any char inside brackets (like glob) [^...] any char not inside brackets (depends on the machine) [!...] any char not inside brackets (depends on the machine) \{m\} exactly m repetitions of previous char \{m,\} at least m reps of prev char \{m,n\} from m to n reps of prev char \< beginning of word \> end of word
script arguments	$0 = name of shell pgm; $1-$n = ordered arguments; $# is number of args passed in; $@ is the arguments as 1 string; $* lists the args separately
grep	prints lines matching a pattern
find	find . -type f -empty ## . is the location (local dir)
head, tail	head -1 #gets top 1 row; tail -1 #gets last 1 rows

Week 3

tr	tr -s ' ' ' ' ## squeezes multiple spaces to one space
sed	sed -e "s/unix/UNIX/" myfile ## -e says use the following cmd, myfile is the file to read, but output to stdout; only changes first occurrence sed -e "s/unix/UNIX/g" myfile ## the g changes multiple occurrences sed can use regex
sort	sorts!
cut	cut -c8 myfile ## output 8th column of each line of myfile cut -c5-7,25- ## outputs columns 5, 6, 7, 25-> of each line of stdin cut -f2 ## outputs the 2nd field of stdin cut -d' ' -f3 ## outputs field 3, fields delimited by one space
paste f1 f2	"joins" tab-separated columns of f1 with tab-separated col's of f2
pipe | and tee	
foreground/background processes	while in foreground process, hit CTRL-Z to suspend jobs: shows current processes fg %1 puts 1 in foreground bg %1 puts 1 in background kill %1 kills 1 kill 2498 kills by process id (find using ps)
ps	show processes ps -u shows user ids too
Command History	!! re-execute last cmd !-n re-execute last command minus n !cmd re-execute last command that started with string cmd history list last 16 commands
lynx	lynx -dump www.google.com ## dumps the webpage rendered as text lynx www.google.com ## tries the diplay the text webpage for use

CPS393 Summary Part 1: Unix/Linux

Week 4

Environment Variables	env; echo “$PWD”
Local Variables	typeset -i X=123; unset x a=$(ls -l | grep “foo” | head -1s)
Quoting, backslash	single quote is more “powerful”, double quotes allow special chars to be evaluated inside; backslash protects the one following char (Also can be used to split single commands to multi-lines)
back-quote	back-quotes execute the command inside (so does $(foo_command))
test	checks a file for readable -r, writable -w, executable -x, is a file -f, is a directory -d compares 2 strings or ints for >, <, = test “abc” = “abc” test 123 -eq 123 can do AND, OR, NOT logic
expr	Evaluates arguments and returns true or false
&&	cmd1 && cmd2 ##execute cmd2 if cmd1 returns true
||	cmd1 || cmd2 ## execute cmd2 if cmd1 returns false
if	if [55 -gt 40]; then echo “55 is grt than 40”; else echo “less than”; fi if test -f fn1; then...; else...; fi if [“`grep $1 f1`”]; then echo “the value $1 was found in f1; fi -- note that back-quotes must be used to evaluate grep, etc

case	mnth=`date +%m` #formats date as mm
	case ${mnth} in
	  01|Jan|January)
	  echo “jan”
	  ;;
	  02|Feb|February)
	  echo “feb”
	  ;;
	  *)
	  echo “whatever”
	  ;;
	esac
	case `expr ${count} \< 100` in
	  1) echo ${count} is less than 100;;
	  0) echo ${count} is more than 99
	esac

for	for var in val1 val2 val3.... do ### do something with var done
while	while [...] do ### do something done
read	read entireline < f1 ## reads first line from f1 read word1 word1 rest < f1 ## reads first word into word, etc ## example of how to read through each line in a file ## unfortunately this piping into a while statement creates a sub-scope ## which means that variables set inside the loop are not still set on the other side cat f1 | \ while read line do ### do something with line done

Week 5

bash -v ./pgm bash -x ./pgm	-v displays the pgm code before execution -x displays the pgm code with values before exec
shift	shifts all arguments down one ($1 gets value from $2, $2 gets value from $3, etc)
xargs	performs a command on a group of things from stdin find . -name “f*” | xargs grep bash ## prints lines from files f* that contain the string bash

functions	repeat_it () {
	echo $1
	}
	
	repeat_it “foo” ## prints “foo”

arguments	
eval	