## Instruction and Cheatsheet

**Please follow the below instructions for running and submitting the programming assignments on bash/awk/sed scripts.**

Your solution code is enclosed within single quotes, thus

Use double quotes whenever possible instead of single quotes. While using double quotes for sed command/script note that the $ symbol has to be escaped with a backslash. Ex: `sed -n "/^start.*end\$/ p"`
In order to use single quotes, use the character sequence `'\''` for every single quote.
Ex: For writing `sed -n '/^start.*end$/ p'`,
write instead `sed -n '\''/^start.*end$/ p'\''`

**Cheat Sheet**



# AWK

## Built-in variables

| Variable | Description |
|---|---|
| ARGC | Number of arguments supplied on the command line (except those that came with -f & -v options) |
| ARGV | Array of command line arguments supplied; indexed from 0 to ARGC-1 |
| ENVIRON | Associative array of environment variables |
| FILENAME | Current filename being processed |
| FNR | Number of the current record, relative to the current file |
| FS | Field separator, can use regex |
| NF | Number of fields in the current record |
| NR | Number of the current record |
| OFMT | Output format for numbers |
| OFS | Output fields separator |
| ORS | Output record separator |
| RS | Record separator |
| RLENGTH | Length of string matched by match() function |
| RSTART | First position in the string matched by match() function |
| SUBSEP | Separator character for array subscripts |
| $0 | Entire input record |
| $n | nth field in the current record |

## Operators

| Type | Operators |
|---|---|
| Assignment | = += -= *= /= %= ^= **= |
| Logical | \|\| && |

| | |
|---|---|
| Algebraic | + - * / % ^ ** |
| Relational | > <= > >= != == |

| Operator | Description |
|---|---|
| expr ? a : b | Conditional expression |
| a in array | Array membership |
| a ~ /regex/ | Regular expression match |
| a  !~ /regex/ | Negation of regular expression match |
| ++ | Increment, both prefix and postfix |
| -- | decrement, both prefix and postfix |
| $ | Field reference |
| Blank | is for concatenation |

## Functions and command

| Type | Commands |
|---|---|
| Arithmetic | atan2 cos exp int log rand sin sqrt srand |
| String | asort asorti gsub index length match split<br>sprintf strtonum sub substr tolower toupper |
| Control Flow | break continue do while exit for if else return |
| Input / Output | close fflush getline next nextline print printf |
| Programming | extension delete function system |
| bit-wise | and compl lshift or rshift xor |

String Functions

Gawk has the following built-in string functions:

asort(s [, d [, how] ]) Return the number of elements in the source array s. Sort the contents of s using gawk's normal rules for comparing values, and replace the indices of the sorted values s with sequential integers starting with 1. If the optional destination array d is specified, first duplicate s into d, and then sort d, leaving the indices of the source array s unchanged.

asorti(s [, d [, how] ])
Return the number of elements in the source array s. The behavior is the same as that of asort(), except that the array indices are used for sorting, not the array values.

gensub(r, s, h [, t]) Search the target string t for matches of the regular expression r. If h is a string beginning with g or G, then replace all matches of r with s. Otherwise, h is a number indicating which match of r to replace. If t is not supplied, use $0 instead.

gsub(r, s [, t]) For each substring matching the regular expression r in the string t, substitute the string s, and return the number of substitutions. If t is not supplied, use $0.

index(s, t) Return the index of the string t in the string s, or zero if t is not present.

length([s]) Return the length of the string s, or the length of $0 if s is not supplied.

match(s, r [, a]) Return the position in s where the regular expression r occurs, or zero if r is not present, and set the values of RSTART and RLENGTH.

patsplit(s, a [, r [, seps] ])
Split the string s into the array a and the separators array seps on the regular expression r, and return the number of fields.

split(s, a [, r [, seps] ])
Split the string s into the array a and the separators array seps on the regular expression r, and return the number of fields. If r is omitted, FS is used instead.

strtonum(str) Examine str, and return its numeric value.

sub(r, s [, t]) Just like gsub(), but replace only the first matching substring. Return either

```
                zero or one.

  substr(s, i [, n])    Return the at most n-character substring
                of s starting at i. If n is omitted, use
                the rest of s.

  tolower(str)      Return a copy of the string str, with all
                the uppercase characters in str translated
                to their corresponding lowercase counter-
                parts. Non-alphabetic characters are left
                unchanged.

  toupper(str)      Return a copy of the string str, with all
                the lowercase characters in str translated
                to their corresponding uppercase counter-
                parts. Non-alphabetic characters are left
                unchanged.
```

### Examples

```
# replace every alphanumeric character to A
awk '{gsub("[[:alnum:]]", "A", $1); print}' file

#
```

## Array

```
arr[index]=value # Assignment
for (var in arr) { print var; } # Iteration through array
delete arr[index] # Delete an array
```

## Conditional and Loops

```
if (a > b) {
print a
}

for (a in array) {
print a
}

for (i=1;i<n;i++) {
print i
}

while (a < n) {
print a
}

do {
print a
} while (a < n)
```

# SED

## Actions

| Keyword | Action |
|---|---|
| p | Print the pattern space |
| d | Delete the pattern space |
| s | Substitute using regex match s/pattern/replacement/g |
| = | Print current input line number, \n |
| # | comment |
| i | Insert above current line |
| a | Append below current line |
| c | Change current line |

## Programming

| Keyword | Action |
|---|---|
| b label | Branch unconditionally to label |
| :label | Specify location of label for branch command |
| N | Add a new line to the pattern space and append next line of input into it. |
| q | Exit sed without processing any more commands or input lines |
| t | label Branch to label only if there was a successful substitution was made |
| T | label Branch to label only if there was no successful substitution was made |
| w filename | Write pattern space to filename |
| x | Exchange the contents of hold and pattern spaces |

# BASH

## Input and Output

## Reading and printing

```
read var
read -a arr # read as array, splitted to multiple elements based on the space

echo hi # print to stdout/screen

echo hi > file # redirect stdout to file
echo error 2> errorlog # redirect stderr to file
```

# Conditional

```
test expression
e.g: test -e file
[ exprn ]
e.g: [ -e file ]
[[ exprn ]]
e.g: [[ $ver == 5.*]]
(( exprn ))
e.g: (( $v ** 2 > 10 ))
command
e.g: wc -l file
pipeline
e.g: who|grep "joy" > /dev/null
! for negation
e.g: ! [ $a = $b ] # note there is a space after !
```

## Conditional Expressions

### Unary file comparisons (test)

```
-e file Check if file exists
-d file Check if file exists and is a directory
-f file Check if file exists and is a file
-r file Check if file exists and is readable
-s file Check if file exists and is not empty
-w file Check if file exists and is writable
-x file Check if file exists and is executable
-O file Check if file exists and is owned by current user
-G file Check if file exists and default group is same as that of current user
```

### String comparison (test)

| expression | description |
|---|---|
| `$str1 = $str2` | Check if str1 is same as str2 |
| `$str1 != $str2` | Check if str1 is not same as str2 |
| `$str1 < $str2` | Check if str1 is less than str2 |
| `$str1 > $str2` | Check if str1 is greater than str2 |
| `-n $str2` | Check if str1 has length greater than zero |
| `-z $str2` | Check if str1 has length of zero |

### Arithmetic comparison

| expression | description |
|---|---|
| `$n1 -eq $n2` | Check if n1 is equal to n2 |
| `$n1 -ge $n2` | Check if n1 is greater than or equal to n2 |
| `$n1 -gt $n2` | Check if n1 is greater than n2 |
| `$n1 -le $n2` | Check if n1 is less than or equal to n2 |
| `$n1 -lt $n2` | Check if n1 is less than n2 |
| `$n1 -ne $n2` | Check if n1 is not equal to n2 |

## Conditional execution

### if-elif-else

```
if condition; then
commands
elif condition; then
commands
else
commands
fi
```

## Case statement

```
case $var in
 op1)
 commandset1;;
 op2 | op3)
 commandset2;;
 op4 | op5 | op6)
 commandset3;;
 *)
```

```
 commandset4;;
esac
```

## Loop

### for do loop

```
for var in list; do
commands
done

for (( i = 0; i < 10; i++ ));do
echo $i
done
```

### while loop

```
while condition; do
commands
done
```

### until do loop

```
until condition; do
commands
done
```

### Select loop

```
echo select a middle one
select i in {1..10}; do
  case $i in
      1 | 2 | 3)
     echo you picked a small one;;
      8 | 9 | 10)
     echo you picked a big one;;
      4 | 5 | 6 | 7)
     echo you picked the right one
     break;;
  esac
done
echo selection completed with $i
```

## Functions

### Definition

```
myfunc() {
commands
}

function myfunc() {
commands
}
```

### Call

```
myfunc
```

## Grep

### Options

| Option | Description |
|---|---|
| -E, --extended-regexp | PATTERNS are extended regular expressions |
| -F, --fixed-strings | PATTERNS are strings |
| -G, --basic-regexp | PATTERNS are basic regular expressions |
| -P, --perl-regexp | PATTERNS are Perl regular expressions |
| -e, --regexp=PATTERNS | use PATTERNS for matching |
| -f, --file=FILE | take PATTERNS from FILE |
| -i, --ignore-case | ignore case distinctions in patterns and data |
| -v, --invert-match | select non-matching lines |
| -m, --max-count=NUM | stop after NUM selected lines |
| -n, --line-number | print line number with output lines |
| -H, --with-filename | print file name with output lines |
| -o, --only-matching | show only nonempty parts of lines that match |
| -r, --recursive | like --directories=recurse |
| -L, --files-without-match | print only names of FILEs with no selected lines |
| -l, --files-with-matches | print only names of FILEs with selected lines |
| -c, --count | print only a count of selected lines per FILE |

_

# Regex

## Special characters (BRE & ERE)

| | Description |
|---|---|
| . | Any single character except null or newline |
| * | Zero or more of the preceding character / expression |
| [ ] | Any of the enclosed characters; hyphen (-) indicates character range |
| ^ | Anchor for beginning of line or negation of enclosed characters |
| $ | Anchor for end of line |
| \ | Escape special characters |

## Special characters (BRE)

| | Description |
|---|---|
| {n,m} | Range of occurances of preceding pattern at least n and utmost m times |
| ( ) | Grouping of regular expressions |

## Special characters (ERE)

| | Description |
|---|---|
| {n,m} | Range of occurances of preceding pattern at least n and utmost m times |
| ( ) | Grouping of regular expressions |
| + | One or more of preceding character / expression |
| ? | Zero or one of preceding character / expression |
| \| | Logical OR over the patterns |

## Character classes

| | description |
|---|---|
| [[:print:]] | Printable |
| [[:blank:]] | Space / Tab |
| [[:alnum:]] | Alphanumeric |
| [[:space:]] | Whitespace |
| [[:alpha:]] | Alphabetic |
| [[:punct:]] | Punctuation |
| [[:lower:]] | Lower case |
| [[:xdigit:]] | Hexadecimal |
| [[:upper:]] | Upper case |
| [[:graph:]] | Non-space |
| [[:digit:]] | Decimal digits |
| [[:cntrl:]] | Control characters |

## Backreferences

\1 through \9
\n matches whatever was matched by nth earlier paranthesized subexpression
A line with two occurances of hello will be matched using: \(hello\).*\1