http://kirschner.med.harvard.edu/xen_bioinfo/

# Xenopus Bioinformatics: AWK+

HMS
Leon Peshkin
pesha@hms.harvard.edu

# Advanced UNIX Topics

## Command line editing

- Control-A (^A): Move cursor to beginning of line. Mnemonic: A is first letter of alphabet
- ^E: **E**nd of line

(^Z was already taken for something else).

- ^D: **D**elete character currently under the cursor.
- ^K: **K**ill (cut) from the cursor to end of line. (Deleted text goes to a clipboard)
- ^Y: **Y**ank (paste) the clipboard text back onto the command line

## File/Directory Permissions

- Every file and directory has an owner (a user) and a group
- `groups lp28` – groups I belong to
- `ls -l` shows each file's owner/group
- `chown`, `chgrp` changes these values

```
botka@portal.77% ls -la
total 64
drwxrwxr-x    2 botka    botka        4096 Feb 25 08:06 ./
drwxrwx---   48 botka    cgradmin    45056 Feb 25 08:02 ../
-rw-rw-r--    1 botka    botka        5332 Feb 25 08:02 moreseqs
-rw-rw-r--    1 botka    botka        1102 Feb 25 08:06 opsd_human.fasta
-rw-rw-r--    1 botka    botka        1247 Feb 25 08:02 seqs
```

## Permissions II

- `ls -l` says who can do what to a file/directory:
  - r: read, w: write (or delete), x: execute a file, see inside a directory
  - categories: **u**ser, **g**roup, **o**ther
- `chmod` changes these values
  - `chmod o+w seqs` (now others can edit the file)
  - `chmod 644 seqs` (magic to set permissions: see chmod man page)

```
botka@portal.77% ls -la
total 64
drwxrwxr-x    2 botka    botka        4096 Feb 25 08:06 ./
drwxrwx---   48 botka    lp28        45056 Feb 25 08:02 ../
-rw-rw-r--    1 botka    botka        5332 Feb 25 08:02 moreseqs
-rw-rw-r--    1 botka    botka        1102 Feb 25 08:06 opsd_human.fasta
-rw-rw-r--    1 botka    botka        1247 Feb 25 08:02 seqs
```

## Environment Variables

- Information about your account
- Preferences for your account
- Locations of databases, files, programs
- `tcsh:`
  - `setenv BLASTDB ~/my_blastdbs`
  - `printenv BLASTDB`
- `bash:`
  - `Set BLASTDB = ~/my_blastdbs`
  - `echo $BLASTDB`

## The UNIX $PATH

- `PATH` is an environment variable set up by the system
- Lists the places where the shell looks for executable files (`ls` is really `/bin/ls`)
- Set automatically, but you can add to it
- Change it in your `.tcshrc/.bashrc` file.
  - `bash: set path=( ~/bin $path )`
  - `tcsh: setenv PATH "~/bin $PATH"`

## Standard Output and Error

`some_long_program > long.out`

- If there's an error, I don't want to wait for the whole program to finish to find out
- So (well-behaved) programs split output:
  - standard output (`stdout`) has regular info
  - standard error (`stdout`) has errors and warnings
- Both go to the screen by default
- > and >> only redirect `stdout` to a file
- >& and >>& will redirect `stdout` AND `stderr`
- `bsub -o` redirects `stdout` and `stderr` to a file

## UNIX Scripting

- UNIX shell has a whole programming language
  - Variables, loops, conditions, etc.
  - Language is slightly different for `bash` vs. `tcsh`
  - Examples are `tcsh` unless otherwise noted

```
portal> foreach i (*seqs)
foreach? echo $i
foreach? grep -c 'WAR' $i
foreach? end
```

## UNIX Scripting II

- Create scripts using text editors:
  - `Emacs, pico` (good for beginners), `vi` (Vim)
- Run scripts by
  - `chmod +x blah.sh`
  - `./blah.sh`
  - Or just `tcsh blah.sh`
- Commands, loops, etc. run as if you typed them in at the command line

```
foreach i (*seqs)
  echo $i
  grep -c 'WAR' $i
end
```

## UNIX Scripting III

- `./myscript a b c`
  - $1 is "a", $2 is "b", $3 is "c"
  - print, compare, etc. the $ variables in script
  - The set command creates normal variables
- Conditions:

```
if ($1 == 1) then
  echo "hi"
else
  echo "bye"
endif
```

- Read tcsh (or bash) man pages for much more

## Login rc Files

- Some scripts automatically run when you login
  - `tcsh: /etc/csh.cshrc, /etc/csh.login, .tcshrc`
  - `bash: /etc/profile, /etc/bashrc, .bashrc`
- These are just regular shell scripts
- Put commands in here that you want to run every time you login

## More Shortcuts: Aliases and Links

- `ln -s ../../some/far/away/file ./here`
  - ln is just like cp, but it makes a link instead
  - more here will more the far away file, etc.
- `alias cdd 'cd some/far/away/dir'`
  - put this in your .tcshrc so you always have it
- alias can also use variables!
  - `alias lastlog 'set lastlog=`ls -dtr /usr/local/adm/log/updatedb/{\!:*}* | tail -n 1`; echo "Most recent \!:* log: $lastlog"; more $lastlog`

## More commands

- /bin has 81 commands
  - And then there's /usr/bin, /usr/local/bin...
- Data manipulation: sort, cut, paste, join, tr
- File filters: sed, awk
- Real programming languages: Perl, Python

## AWK Program

```
gawk 'BEGIN{RS="//";FS="AUTHORS|TITLE|JOURNAL"}
{n=split($2,a," ");c[a[n]]++}END{for (i in c) print i"\t"c[i]}'
cDNA-GenBank-format-XI-MB-UNIX.txt | sort -rnk2 | more
```

## Running An AWK Program

**awk [-Ffs] [*'program'*|-f progfile] [datafile...]**

| | |
|---|---|
| fs | field separator |
| *'program'* | AWK program entered on cmd line |
| progfile | text file containing AWK program |
| datafile | data you want to pass through your AWK program ("-" is stdin) |

```
gb|AAN86046.2| rhodopsin [Sminthopsis crassicaudata]              617   e-177
gb|AAP35089.1|AF425072_1 RH1 opsin [Oncorhynchus mykiss]   553   e-158
dbj|BAC76806.1| green-sensitive opsin 1 [Cyprinus carpio]          508   e-144
dbj|BAC76807.1| green-sensitive opsin 2 [Cyprinus carpio]          500   e-142
gb|AAP35093.1|AF425076_1 RH2 opsin [Oncorhynchus mykiss]   441   e-124
gb|AAO38746.1| green rod opsin [Xenopus laevis]                356   1e-98
gb|AAP35092.1|AF425075_1 SWS2 opsin [Oncorhynchus mykiss]          322   2e-88
gb|AAP35091.1|AF425074_1 SWS1 opsin [Oncorhynchus mykiss]          292   3e-79
gb|AAP37944.1| short wave-sensitive opsin 1 [Macropus eugenii]       287   8e-78
gb|AAP30082.1| opsin [Parus caeruleus]                            284   7e-77
gb|AAP37945.1| medium wave-sensitive opsin 1 [Macropus eugenii]      266   2e-71
gb|AAP35090.1|AF425073_1 LWS opsin [Oncorhynchus mykiss]           255   4e-68
gb|AAP30087.1| opsin [Luscinia svecica]                           247   1e-65
gb|AAM77793.1| vertebrate ancient opsin long isoform [Rutilus ru...  243   1e-64
gb|AAP30084.1| opsin [Parus palustris]                            240   1e-63
gb|AAP30085.1| opsin [Euplectes afer]                             238   6e-63
gb|AAM77794.1| vertebrate ancient opsin short isoform [Rutilus r...  232   2e-61
gb|AAP30088.1| opsin [Luscinia calliope]                          231   5e-61
gb|AAP30083.1| opsin [Parus major]                                224   9e-59
gb|AAP30086.1| opsin [Euplectes orix]                             220   1e-57
```

## Internal Variables

```
$ cat data1
one
two two
three three three oops
four four four four
$
$ awk '{print NR,$0,"FIELD COUNT:",NF}' data1
1 one FIELD COUNT: 1
2 two two FIELD COUNT: 2
3 three three three oops FIELD COUNT: 4
4 four four four four FIELD COUNT: 4
```

## Internal Variables Continued

**NF** field count for current record
**NR** count of records read so far
**FNR** count of records read from current file

**FS** input field separator
**RS** input record separator
**OFS** output field separator
**ORS** output record separator

**$0** entire input line
**$1** first field, $2 is second field ....

## Even More About Internal Variables

**NF** is an integer
(the number of fields in this record)

**$NF** is the contents of the last
field in the record

**$NR** the contents of the NRth field in
this record

```
$ cat data2
a b c d e
f g h i j
k l m n o p
q r s t u
v w x y z
$
$ awk '{print $NR}' data2
a
g
m
t
z
```

```
$ cat data2
a b c d e
f g h i j
k l m n o p
q r s t u
v w x y z
$
$ awk '{print $NF}' data2
e
j
p
u
z
```

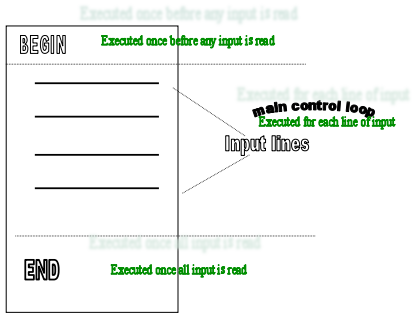
## AWK Program Structure

**PATTERN {action}**
**PATTERN {action}**
.
.
.

## PATTERNS

**BEGIN**
**END**
**/regular_expression/**
**$1 ~ /regular_expression/**
**$1 !~ /regular_expression/**
**NF != 3**
**$2 == 5**
**$1 == “literal_string”**
**$2 >= 4 || $3 <= 20**

## awk




## Regular Expressions

| | |
|---|---|
| ^ | beginning of line |
| $ | end of line |
| . | any single character |
| [ ] | character class |
| \| | alternation |
| * | closure (zero or more) |
| + | positive closure (one or more) |
| ? | zero or one |
| ( ) | grouping |
| \ | escapes meaning of meta character |

## Regular Expression Examples

**(BOZO|COOKIE|WIZZO)(UP|DOWN)(TO|YESTER)DAY**

BOZO UP TODAY
COOKIE DOWN YESTERDAY
COOKIE UP YESTERDAY

**(AB)+c**

ABc
ABABc
ABABABc

## /^(\+|-)?[0-9]+\.?[0-9]*$/

!

## Pattern Ranges

**/BOZO/,/PROFANDY/**

***BOZO tricks COOKIE***
***WIZZO botches magic trick***
***PROFANDY plays music***
**PROFANDY gets cream pie**
*BOZO chases COOKIE with pie*
*BOZO sings to audience*
*PROFANDY talks with GOLLY*
**WIZZO performs magic trick**
**WIZZO receives cream pie**

## Actions

- Output
- Data Manipulation
- Flow Control

## Output

**print** **basic output**

**quoted strings are output verbatim**
**comma-separated arguments are output with the OFS between them**

**printf** **formatted output (works just like C)**

**first argument is format string**
**other arguments are the values to substitute**

## print

one two three

```
BEGIN       { OFS="|"}
            { print $1 $2 "test",$3,"wow"}
```

onetwotest|three|wow

## printf

one two three

```
BEGIN{ OFS="|"}
      { printf("%s%s%s%s\n",$1,$3,OFS,"wow"}
```

onethree|wow

## printf format strings

```
                                |1234567890|
printf("|%c|",100)              |d|
printf("|%5d",100)              |  100|
printf("|%7.2f|,100.5)         |  100.5|
printf("|%s|","MySystm")       |MySystm|
printf("|%-10s|","MySystm")    |MySystm   |
printf("|%10s|","MySystm")     |   MySystm|
printf("|%5s|","MySystm")          |MySystm|
                                |1234567890|
```

## Data Manipulation

- Built-In Functions
  - String
  - Numeric
- Operators

## Built-In Functions (String)

**gsub(r,s,t)** **substitue s for r in string t**
**index(s,t)** **return first postition of t in s**
**length(s)** **number of characters in string s**
**split(s,a,fs)** **split s into array a on field separator fs; returns field count**
**sub(r,s,t)** **substitute s for the leftmost longest substring of t matched by r**
**substr(s,p,n)** **return substring of s of length n starting at position p**

## XXXXYYYYMMDDWWWWWW

```
{
print substr($0,8,4)substr($0,4,4)
}
```

**MMDDYYYY**

## Built-In Functions (Numeric)

| | |
|---|---|
| **cos(x)** | **returns the cosine of x radians** |
| **exp(x)** | **exponential (exp(1) returns e)** |
| **int(x)** | **returns integer portion of x** |
| **log(x)** | **natural logarithm of x** |
| **rand(x)** | **random number ( 0 <= r < 1)** |
| **sin(x)** | **returns the sine of x radians** |
| **sqrt(x)** | **returns the square root of x** |

## Operators

| | |
|---|---|
| **Assignment** | **= += -= *= /= %= ^=** |
| | **( y *= 2 is y = y * 2 )** |
| **Conditional** | **?:** |
| **logical OR** | **\|\|** |
| **logical AND** | **&&** |
| **match** | **~ !~** |
| **relational** | **< <= == != >= >** |
| **add,subtract** | **+ -** |
| **multiply,divide** | *** /** |
| **mod** | **%** |
| **logical NOT** | **! (!$1 - 1 if $1 is zero or null)** |
| **exponentiation** | **^ ( x ^ y = $x^y$ )** |
| **increment,decrement** | **++ --** |
| **grouping** | **( )** |
| **field** | **$ ( $(n+1) is $n^{th}$+1 field )** |
| **array membership** | **in** |

## $A^2 + B^2 = C^2$

```
{
        print sqrt( ($1 * $1) + ($2 * $2) )
}
```

**or**

```
{
        print sqrt( ($1 ^ 2) + ($2 ^ 2) )
}
```

## Assignment

```
orchestra> cat data
a 10 5 13
b 20 4 21
c 30 3 18
d 40 2 66
orchestra> cat pres.awk
      { $2=$2+$3 }
$2 > $4 { print $0 }
orchestra> awk -f pres.awk data
a 15 5 13
b 24 4 21
c 33 3 18
```

## Flow-Control

**if (expression) statement [else statement]**
**while (expression) statement**
**for (expression; expression; expression) statement**
**do statement while ( expression )**

| | |
|---|---|
| **break** | **-** breaks innermost while,for,do |
| **continue** | **-** next iteration of innnermost while,for,do |
| **next** | **-** next iteration of main input loop |

**exit [expression]**

```
BEGIN  {
       maxwidth=79
       }

       {
       if ( $1 <= maxwidth )
            {
            for (i=$1;i > 0; i--) printf("*")
            printf("\n")
            }
       else
            {
            printf("Value %d at line %d > %d\n",$1,NR,maxwidth)
            }
       }
```

# Command Line Variables

```
awk -f prog.awk a=- c=2 d=3 data0 a=k b=y c=x data1 data2

        data0  a=-    b=""    c=2    d=3
        data1  a=k    b=y     c=x    d=3
        data2  a=k    b=y     c=x    d=3
```

# Arrays

- Declaration not necessary
- Subscripts are strings
- Multi-dimensional obtained by subscript concatenation
- Element Occurrance ("in")

# Arrays (reverse.gawk)

```
# Output lines of file in reverse order
       {
       array[NR]=$0
       }

END    {
       for (i=NR;i>0;i--) print array[i]
       }
```

# Arrays (primaries.gawk)

```
BEGIN  {
       primaries["red"]=1
       primaries["blue"]=2
       primaries["yellow"]=3
       }

       {
       if ( $1 in primaries ) print $1,"is a primary color"
       else print $1,"is NOT a primary color"
       }
```

# Arrays Continued

```
       { cntarray[$3]=cntarray[$3]+1 }
END    { for ( i in cntarray ) print i,
  cntarray[i]}

       totals[$1","hour","$3]=
```

# Multiple-Line Records

**DATA(mlr.fasta)**

```
>prot1
RKRKRKRKRKRKR
KRKRKRKRKRKRKRKR

>prot100
SVLIVLSISLIVSLIVSLIVLSIV
SVLISVLISVLISVLISVLSIV
SVLISLIVSLIVSLVI

>prot10000
EVILEVILEVILEVILEEVILEILEVIEL
```

**Program(mlr.awk)**

```
BEGIN   {
        RS=""
        FS="\n"
        }

        {
        print $1
        }
```

**RUN**

```
$ awk -f mlr.awk mlr.fasta
>prot1
>prot100
>prot10000
$
```

# CONCLUSION

- experiment
- walk before you run
- you might find AWK useful for:
  - making sure every record of a file has the same field count
  - manipulating numeric information
  - creating reports from raw data
  - gathering specific information from reports
  - data conversions

# Useful AWK One-Liners

```
{ print $NF }       # Last field on every line
```

---

```
        { if ( NF > mfields ) mfields = NF }
END     { print "MOST FIELDS =",mfields}
```

---

```
        { if ( length($0) > wideline ) wideline = length($0) }
END     { print "WIDEST LINE =",wideline }
```

---

```
{ print $($1) } # print the field referenced by first field
```

---

```
NF > 0 { print $0 } # removes blank lines
```

---

```
{ print NR":",$0 } # numbers the lines in a file
```

# More Useful AWK One-Liners

```
NF != 5 { print NR,NF,$0 } # If a line does not have exactly 5 fields
                           # print the line number, the number of
                           # fields found and the line itself
```

---

```
        { total=total + $2 }
END     { print total }     # total of field two
```

---

```
$3 > max        { max = $3 ; maxline = $0 }
END             { print maxline } # find the maximum value for
                                  # field three and print the line
```

---

```
{
 for (i=NF;i>0;i--) printf("%s ",$i)
 printf("\n")
} # reverse the order of fields on each line
```

# Still More AWK One-Liners

```
####### potential orphaned processes
ps -ef | awk '$3 == 1 { print $0 }'
```

---

```
####### Center Lines of Text (need to pass width)
        {
         format=sprintf("%%%ds", width/2 - length($0)/2)
        printf(format"%s\n", " ", $0)
        }
```

---

```
####### total each input line
{
total=0
for (i=NF;i>0;i--) total=total+$i
print total
}
```

# REFERENCES


- The AWK Programming Language
  - Alfred V. Aho
  - Brian W. Kernighan
  - Peter J. Weinberger
- SED & AWK
  - Dale Dougherty
- Mastering Regular Expressions
  - Jeffrey E. F. Friedl