

Programming Linux environment

Variables

Basic Input/output

Permissions

Download data files

Environment variables

- How does the terminal know **where to look** for executables? (e.g. how does it know `ls` is in `/usr/bin`?)
- A **variable** is a word that represents/contains a value or string. Environment variables describe your system
- `MY_SYSTEM=Ubuntu-Linux`

Set of variables: environment

- Try this command

env

- What do you see?
- Look for PATH variable

PATH=/usr/bin:/usr/local/bin:/usr/sbin

- Try the command:

echo \$PATH

PATH environment variable

- `echo $PATH`
- `/home/toni/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games`
- Path contains a list of directories separated by the symbol “:”

Installing an application in a local folder

- You **copy** the executable to one of the folders in PATH

```
$ sudo cp /home/toni/Downloads/app /usr/local/bin
```

```
$ sudo chmod +x /usr/local/bin/app
```

- You **create a sym(bolic) link** to an executable in the one of the folders in PATH
- You **add a directory to the PATH variable**

```
Export PATH=/home/masterbio/myapp/app:$PATH
```

Example: installing Blast

Steps to have a local Blast in your home

- Download blast and md5 files
- Check file integrity
- Unpack file
- Copy blast to installation folder (/home/toni/myblast)
- Check permissions and execution of myblast
- Create a symlink in home folder
- Add new blast tool to \$PATH variable

Blast download links:

<ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.5.0+-src.tar.gz>

<ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.5.0+-src.tar.gz.md5>

Installing apps in the system

- What if a new version comes up?
- How to make application available for other users?
- What if I need shared system libraries for my application?

```
sudo apt-get update
```

```
sudo apt-get install blast
```

Variables and scripts

- You can write a list of commands one after the other in a **text file**, and let bash execute it.
- Let's try!
 - Create a file in your home called “**space_left**” with text editor
 - Enter two following bash commands in this file:

```
df -h .  
du -sh */
```
 - Then write at the console: **bash ./space_left**

Bash script

- Simple text files become Bash scripts when adding a **shebang** line as first line
- Shebang states which program should read and execute the text file

```
#!/bin/bash
```

```
#!/usr/bin/perl
```

```
#!/usr/bin/python
```

WORK!

- Write a script with two variables and two messages
- Use the command echo to print message, name and the variable USER

```
message="hello"  
name="toni"  
echo "$message, $USER !"  
echo "my name is $name"
```

WORK!

- Modify your script to:
 - save the date of today in a variable
 - print a hello message with the date
 - use **read** command to get the name of the user
- ```
read -p "please tell me your name: " user_name
```
- print a hello message with the user name you just read

# File permissions

- `chown user:group filename`
- `chmod [ugo][+-][rwx] filename`
- `chmod [0-7][0-7][0-7] filename`
  - 1 stands for execute
  - 2 stands for write
  - 4 stands for read
  - any number from 0 to 7 is a unique combination of 1, 2 and 4.

# File permissions

- Can you run your script?
- What's missing?

# Script arguments

- We can pass on **arguments** to our scripts: they are subsequently stored in variables called \$1, \$2, \$3,...
- **Edit a new file called 'arguments.sh'** with following contents (be aware of the “)

```
#!/bin/bash
firstArg=$1
secondArg=$2
echo "You have entered \"$firstArg\" and
 \"$secondArg\""
```



# Script arguments

- Make your script executable

```
chmod +x arguments.sh
```

- Run your script

```
./arguments.sh first second
```

# WORK!

- Modify your hello script to:
  - get the user name from an argument
  - print the argument received from the user
  - make your script executable

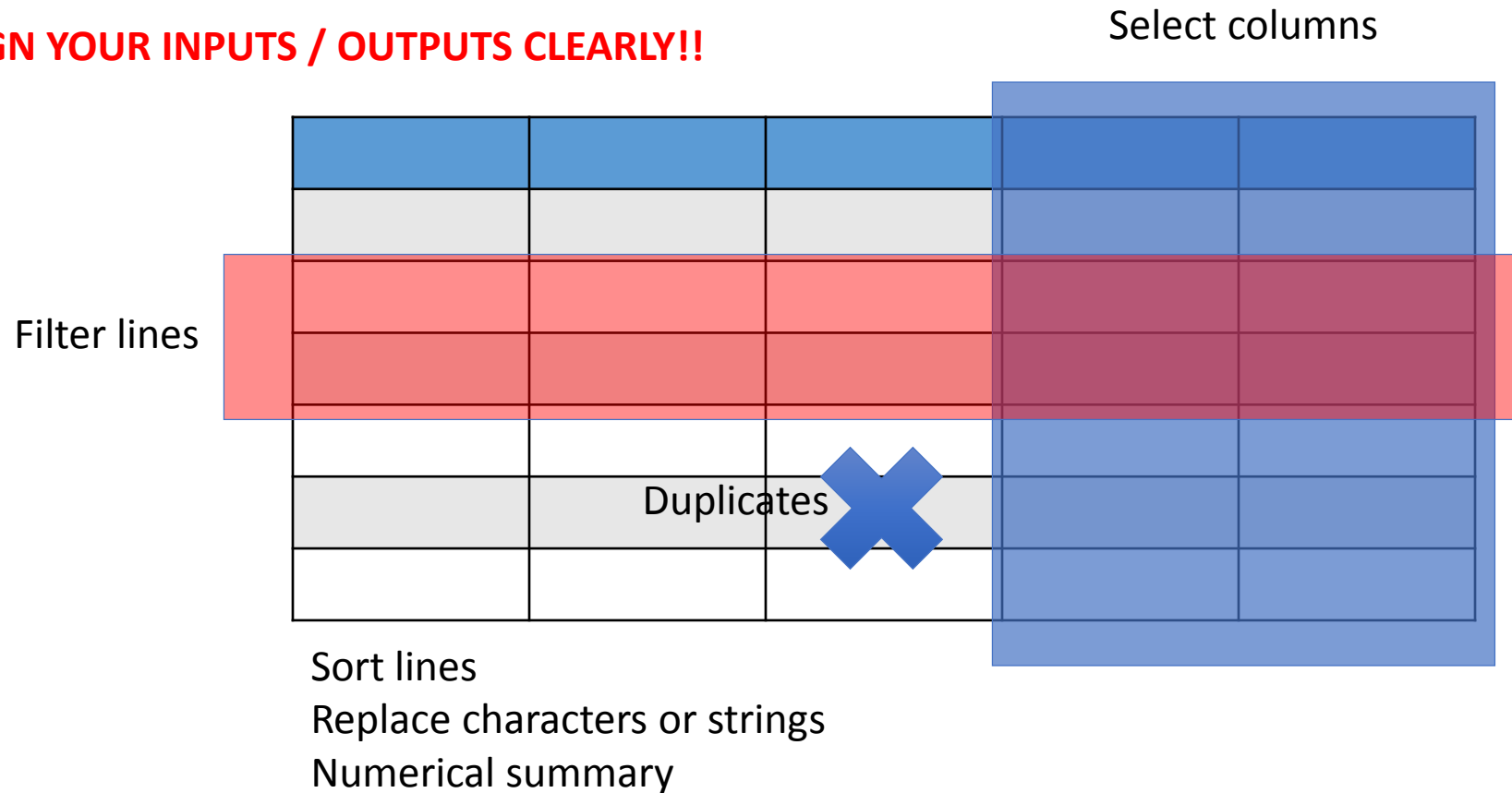
# Linux text tools and scripting

# General text manipulation

- Matrix of text
  - Each element is a line
  - Each feature is a column
- Typical operations
  - Count, filter, select, summarize, search, ...

# Script=Designing filters

**DESIGN YOUR INPUTS / OUTPUTS CLEARLY!!**



# Choose your weapon

- UNIX has an extensive toolkit for **text** analysis:
  - **Extraction**: head, tail, grep, awk
  - **Reporting**: wc
  - **Manipulation**: sort, tr, sed
- Complex text parsing needs more specific programming: **Python, Go, Ruby, ...**

# Output redirection

- Output from script must be saved in a new file
- Example: sort list of search results by score
- The output of a program can be saved to a file

```
ls -lR /home/toni > /tmp/ls-dump.txt
less /tmp/ls-dump.txt
```

# Not overwriting output

- If you write to a file with “>”, contents are replaced

- You can append to a file using >>

```
echo “hello” >> /tmp/hello-test.txt
```

```
echo “world” >> /tmp/hello-test.txt
```

```
cat /tmp/hello-test.txt
```

```
wc /tmp/hello-test.txt
```

- What is **wc** doing?



# Doing many things at the same time

- Dump the output of an **ls** command to a file
- Open the generated text and read it
- Remove the file

```
ls -lR /home/toni/dataset1 > /tmp/ls-dump.txt
cat /tmp/ls-dump.txt
rm /tmp/ls-dump.txt
```

# Too much typing!!



# Building pipelines of commands

- A pipeline reads a file and then processes it using the output of a tool as the input to the next
- Dump the output of an `ls` command to a file
- Open the generated text and read it
- Remove the file

```
ls -lR | less
```

BEWARE!->Output of `ls -lR` is the input of `less`

# WORK

- Write a pipeline that counts the lines of the result of an `ls -lR` command

# Linux text processing tools

Read a text file to modify its contents

- sort
- uniq
- cut
- grep

# Sorting text

sort

-n: sort numerically

-f: case insensitive

-r: reverse sort order

-ts: use s as field separator (instead of space)

-kn: sort on the n-th field (1 being the first field)

- `cat months.txt`
- `sort months.txt`

# WORK!

- Sort months by the second column.
- Is it working? How can you sort numerically?
- Sort months by the second column largest values first

# Columns work with cut

Try these commands and understand their output

- `cut -f1 months.txt`
- `cut -d ' ' -f1 months.txt`
- `cut -d ' ' -f1,2 months.txt`
- `cut -d ' ' -f1-3 months.txt`



# WORK!

- Create a file called `seasons.txt` where you write the result of extracting the third column
- Sort this new `seasons.txt` file
- now use `sort -u` option to sort the file. What difference you see in the output?
- Now extract 3<sup>rd</sup> column and sort the file using a pipeline

# WORK!

- Check the result of adding these commands to the previous cut pipelines
- `sort -u`
- `sort | uniq`
- `sort | uniq -c`

# Grep: search and you shall find

- **grep** extracts lines that match a string

`grep [options] regular-expression [file]`

- The input file is read line by line
- If the line matches the criteria defined by regular expression the line is written to the standard output
- Example: `grep winter months.txt`

# Linux tools: grep

Try these grep operations:

- `grep "december" months.txt`
- `grep "12 winter" months.*`
- `grep "7 winter" months.*`
- `grep -n "12 winter" months.*`

# Filtering and processing input with bash scripts

# Welcome to scripting

- Make sure you understand the process before you start: where are the input files, the results?
- Which operations you need to do: design
- Which tools are you going to use: find the right options
- Work with each tool individually and then join inputs and outputs: prototype and test!
- Write and document your script for the next user after you
- There is more than one way to do it!

# Grep example

- A GFF file contains genome annotation info
- <http://www.sequenceontology.org/gff3.shtml>
- Objective: get all lines form locus Os01g01070 from all.gff3

# Example GFF

| seqId | source     | type | Start | End   |   |   |   | Attributes (ID of feature) |
|-------|------------|------|-------|-------|---|---|---|----------------------------|
| Chr1  | MSU_osa1rt | gene | 2903  | 10817 | . | + | . | ID=LOC_OS01g01010          |
| Chr1  | MSU_osa1rt | exon | 2903  | 3268  | . | + | . | ID=LOC_OS01g01010          |

```
grep LOC_Os01g01070 all.gff3
```



# Grep options

- i: ignore case
- v: inverse, shows lines that do not match
- l: list, show the name of the files that contain a match
- n: shows line number of the match

# Grep example

- Use article-large.csv to answer the question:
- How many of those articles are about Linux?

```
grep -i "Linux" articles-large.csv
```

# Regular expressions

- Formal way of describing sets of strings

. = any character

^= beginning of the line

\$= end of the line

+= any string

[]= set of characters

[1-9]= any number between 1 and 9

Example, chromosome 1 to chromosome5 data:

```
grep chr[1-5] all.gff3
```

# Regular expressions

Example, find chromosome 1 to chromosome5 data:

```
grep chr[1-5] all.gff3
```

- chr1: OK
- chr2: OK
- chr9: NO MATCH
- chr20: NO MATCH

# Filter mRNA structures from chr1 and only on the + strand

```
egrep '^chr1' all.gff3
```

All lines that...

^ : Start at the beginning of the line  
chr1: then have "chr1" string

# Filter mRNA structures from chr1 and only on the + strand

```
egrep '^chr1' all.gff3
egrep '^chr1.+' all.gff3
```

. :matches any character

.+: matches any positive number of characters

->Any features in chr1

# Filter mRNA structures from chr1 and only on the + strand

```
egrep '^chr1.+\\+' all.gff3
```

\\+ : Match only those lines that contain a "+" symbol

->only structures on the + strand

Filter mRNA structures from chr1 and only on the + strand

```
egrep '^chr1.+\+' all.gff3 >/tmp/chr1-test.txt
```

Output file in chr1-test.txt



# Linux tool: word count

- `wc [options] file`
  - c: show number of characters
  - w: show number of words
  - l: show number of lines

```
wc -l months.txt
```

# How many mRNA entries on chr1?

- First, filter chr1 lines from input

```
grep chr1 all.gff3
```

- Then, count the number of lines

```
grep chr1 all.gff3 | wc -l
```

# WORK

- How many articles about Linux in articles-large.csv?

# GFF WORK example

| seqId | source     | type | Start | End   |   |   |   | Attributes (ID of feature) |
|-------|------------|------|-------|-------|---|---|---|----------------------------|
| Chr1  | MSU_osa1rt | gene | 2903  | 10817 | . | + | . | ID=LOC_OS01g01010          |
| Chr1  | MSU_osa1rt | exon | 2903  | 3268  | . | + | . | ID=LOC_OS01g01010          |

```
grep LOC_Os01g01070 all.gff3
```

# WORK!

1. How would you change the last grep so that you match all lines except those at chromosome 1?
2. How many genes are in the gff file format? Use a pipeline to solve this question
3. How many different genes?
4. Sort gff file by chromosome and by source
5. Sort mRNA by chromosome number and then by starting position

# WORK!

1. How would you change the last grep so that you match all lines except those at chromosome 1?

```
grep -v chr1 all.gff3 | wc -l
```

# WORK!

2. How many genes are in the gff file format?  
Use a pipeline to solve this question

```
cat file.gff | wc -l
```

```
cut -f3 -d ' ' file.gff | wc -l
```

# WORK!

3. How many different genes?

```
cut -f9 file.gff|sort|uniq|wc -l
```



# WORK!

4. Sort gff file by chromosome and by source

```
sort -k1 -k2 file.gff
```

Sort gff file by chromosome number and then by starting position

```
cut -f1 -f4 file.gff|sort -k1 -k2 >/tmp/chr_test.txt
```

# Awk: filter and modify

- Use awk to extract specific fields from a file
- Then, do manipulations or calculations on the extracted fields

```
awk -F delimiter '{ print $X }'
```

delimiter is the field separator (default is space)

\$X is the field number

\$0: complete line

\$1, \$2, \$3, : first, second, third field

\$NF: last field

<http://www.grymoire.com/Unix/Awk.html>

# Get gene positions from gff example

- Get first 10 gene positions from example

- Get starting position from example:

```
awk '{print $4}' file.gff
```

- Get only first line

```
awk '{print $4}' file.gff | head -1
```

- Get first 15 lines for genes in chr1

```
awk '{if($1=="chr1") print $4;}' file.gff | head -15
```

# GFF WORK example

| seqId | source     | type | Start | End   |   |   |   | Attributes (ID of feature) |
|-------|------------|------|-------|-------|---|---|---|----------------------------|
| Chr1  | MSU_osa1rt | gene | 2903  | 10817 | . | + | . | ID=LOC_OS01g01010          |
| Chr1  | MSU_osa1rt | exon | 2903  | 3268  | . | + | . | ID=LOC_OS01g01010          |

# Use awk to convert formats

- Convert GFF format to a personalized: chr, gene id, positions

## Input:

```
Chr1 source type pos1 pos2 aux strand aux gene_id
Chr1 MSU_osalrt gene 2903 10817 . + . ID=LOC_OS01g01010
```

## Output:

```
Chromosome geneid pos1 pos2
Chr1 ID=LOC_OS01g01010 2903 10817
```

# Use awk to convert formats

- Convert GFF format to personalized output

Input:

```
Chr1 MSU_osa1rt gene 2903 10817 . + . ID=LOC_OS01g01010
```

Output:

```
Chr1 ID=LOC_OS01g01010 2903 10817
```

```
awk '{print $1"\t"$9"\t"$4"\t"$5}' file.gff
```

# Use awk to convert formats

```
awk 'BEGIN {print "chromosome\tgeneid\tpos1\tpos2" }
 {print $1"\t"$9"\t"$4"\t"$5}
 {END print "End of report----"} '
file.gff
```

# WORK!

- How many unique authors are on that list?
- How many articles did each author write?

```
Post date,Content type,Author,Title,Comment count,Path,Tags,Word count
31 Jan 2017,Article,Scott Nesbitt,Book review: Ours to Hack and to
Own,0,/article/17/1/review-book-ours-to-hack-and-own,Books,660
```



# Sed: changing text on the fly

- Sed (stream editor) can be used to make changes in lines of text
- <http://www.grymoire.com/Unix/Sed.html>
- Substitution tool is very used

```
sed -e 's/r1/s1/' file
```

- s: substitute command
- /: separator
- r1: regular expression to be replaced
- s1: text that will replace regular expression match

# Example

- Write a file with text editor with the lines

```
Hello,hello,hello
```

```
Hello,hello,hello
```

- Now let's apply sed to translate commas into semi-colons

```
sed -e 's/,/;/' hello.txt >hello-sed.txt
```

- Now try this:

```
sed -e 's/,/;/g' hello.txt >hello-sed.txt
```

# More cut options

- Use cut to extract fields from text files

- By extracting fixed text sizes

```
cut -c <fields> file
```

- <fields> can be
    - N: n-th element
    - N-M: from N to M
    - N- : from N element on
    - -M: until the M element

# Example: cutting text columns

- Create a acgt.txt file with the text

ACGTACGTacgtACGTACGT

- To extract a range of characters:

```
cut -c 9-12 acgt.txt
```

```
cut -c 9- acgt.txt
```

```
cut -c -12 acgt.txt
```

# uniq to find the needle in the haystack

Use uniq to:

- Eliminate duplicate lines: u
- Display unique lines: u
- Show and count duplicate lines: d
- **INPUT MUST BE SORTED!**

# Duplicate example

- `ls -l /tmp`
- `ls -l /tmp | awk '{print $3}' | sort | uniq`

# Display unique or duplicate lines

- Get lines that appear only once: -u
- Get lines that appear more than once: -d
- Get the count of the lines: -c
- Example

```
ls -l /tmp | awk '{print $3}' | sort | uniq -d
```

# WORK!

- Create a new list of january 20 articles
- Extract the authors of those articles with the number of words each author wrote

Post date,Content type,Author,Title,Comment count,Path,Tags,Word count

31 Jan 2017,Article,Scott Nesbitt,Book review: Ours to Hack and to Own,0,/article/17/1/review-book-ours-to-hack-and-own,Books,660



# Taking decisions with *if*

```
if [expresion1];
then
 expresion2
fi
```

```
V1="foo"
V2="foo2"
if ["$V1" = "$V2"]; then
 echo "TRUE!"
else
 echo "FALSE!"
fi
```

# *if* conditions

## Operator

## Description

|                                    |                                                                         |
|------------------------------------|-------------------------------------------------------------------------|
| <code>! EXPRESSION</code>          | The <code>EXPRESSION</code> is false.                                   |
| <code>-n STRING</code>             | The length of <code>STRING</code> is greater than zero.                 |
| <code>-z STRING</code>             | The length of <code>STRING</code> is zero (ie it is empty).             |
| <code>STRING1 = STRING2</code>     | <code>STRING1</code> is equal to <code>STRING2</code>                   |
| <code>STRING1 != STRING2</code>    | <code>STRING1</code> is not equal to <code>STRING2</code>               |
| <code>INTEGER1 -eq INTEGER2</code> | <code>INTEGER1</code> is numerically equal to <code>INTEGER2</code>     |
| <code>INTEGER1 -gt INTEGER2</code> | <code>INTEGER1</code> is numerically greater than <code>INTEGER2</code> |
| <code>INTEGER1 -lt INTEGER2</code> | <code>INTEGER1</code> is numerically less than <code>INTEGER2</code>    |
| <code>-d FILE</code>               | <code>FILE</code> exists and is a directory.                            |
| <code>-e FILE</code>               | <code>FILE</code> exists.                                               |
| <code>-r FILE</code>               | <code>FILE</code> exists and the read permission is granted.            |
| <code>-s FILE</code>               | <code>FILE</code> exists and it's size is greater than zero             |
| <code>-w FILE</code>               | <code>FILE</code> exists and the write permission is granted.           |
| <code>-x FILE</code>               | <code>FILE</code> exists and the execute permission is granted          |

# WORK!

Open your hello\_script.bash

check that the user name parameter is not empty using *if*

```
#!/bin/bash
firstArg=$1
secondArg=$2
echo "You have entered \"$firstArg\" and
 \"$secondArg\""
```

# Doing repetitive work with *for*

```
for i in 1 2 3;
do
 echo "$i "
done
```

# Repetitive work with *for*

listing files from a folder

```
files = `ls *.txt`
for file in $files
do
 cat $file >> Output.txt
done
```

## repetitive work with *for*

```
for filename in `ls *.gz`
do echo $filename
```

Decompressing on the fly!

# WORK!

```
for filename in `ls *.gz`
do echo $filename
```

Now modify the script to print the first 10 lines of each file in the compressed tarball

# WORK! Plant gene data set

- What plants systems contain a Smell gene?
- How many plant systems contain a Color gene?
- What genes are in common between apple and pear?
- Which are specific to each of them?
- How many genes are in common to all three plant systems?



# WORK! Plant gene data set

- What plants systems contain a Smell gene?

```
grep Smell *.genes
```

- How many plant systems contain a Color gene?

```
grep Color *.genes | cut -f1 -d '.' | uniq -c
```

- How many genes are in common to all three plant systems?

```
cat *.genes | sort | uniq -c | grep " 3 " | wc -l
```

# Pattern matching in linux

Regular expressions in

grep

awk

# Grep regular expressions

```
egrep `^chr1.*\+` file.gff
```

? The preceding item is matched at most once

\* Item will be matched zero or more times

+ Item will be matched one or more times

{n} Item is matched exactly n times

{n,} Item is matched n or more times

{,m} Item is matched at most m times

{n,m} Item is matched at least n times, but not more than m times.

# Grep when finding words

```
grep -i "chr" file.gff
```

- chr1 MATCH
- chr2 MATCH
- chromosome1 MATCH

```
grep -iw "chr1" file.gff
```

- chr1 MATCH
- chr12 NO MATCH
- chromosome1 NO MATCH

# Use grep to find unique number of genes

| seqId | source     | type | Start | End   |   |   |   | Attributes (ID of feature) |
|-------|------------|------|-------|-------|---|---|---|----------------------------|
| chr1  | MSU_osa1rt | gene | 2903  | 10817 | . | + | . | ID=LOC_OS01g01010; ID=...  |
| chr1  | MSU_osa1rt | exon | 2903  | 3268  | . | + | . | ID=LOC_OS01g01013          |

Type must be gene

Must look for non-repeated ID in attributes

Use grep to find unique number of genes gff3

```
cut -f3,9 all.gff3 |
grep '^gene' |
cut -f2 |
cut -f1 -d ';' |
sort |
uniq |
wc -l
```

# AWK syntax and internal variables

```
awk '/search pattern/ {Actions}' input_file
```

The internal variables that awk uses are:

- **FILENAME**: References the current input file.
- **NF**: The number of fields in the current record.
- **NR**: The number of the current record.
- **OFS**: The field separator for the outputted data. By default, this is set to whitespace.
- **ORS**: The record separator for the outputted data. By default, this is a newline character.
- **RS**: The record separator used to distinguish separate records in the input file. By default, this is a newline character.

Find the posts with more than 1000 words

```
awk -F, ' $8 >1000 ' article.csv
```



# Find articles with Linux content

```
awk -F, ' $7 ~/Linux/' articles.csv
```

# Print number articles about Linux

- Check if the content is Linux. If it is yes, in the Action, just increment the count variable, which was initialized with zero in the BEGIN section.

```
$ awk 'BEGIN { count=0;}
 -F, $4 ~ /Linux/ { count++; }
 END { print "Number of Linux papers=",count;}' article.csv
```

Use awk to find unique number of genes

# Use awk to find unique number of genes

```
Awk '{if($3=="gene") print $NF;}' all.gff3 |
cut -f1 -d ';' |
sort |
uniq |
wc -l
```

# awk field separator

- `echo 'foo:123:bar:789' | awk -F: '{print $NF-1}'`
- How would you use awk to separate strings from numbers?
  - Echo 'Sample123string567with89numbers'

# awk field separator

- `echo 'foo:123:bar:789' | awk -F: '{print $NF-1}'`
- How would you use awk to separate strings from numbers?
  - `echo 'Sample123string567with89numbers'`

```
echo 'Sample123string567with89numbers' |
awk -F'[0-9]+' '{print$2}'
```

# Regular expressions with awk

- Catch all lines containing Linux

```
awk '/Linux/' articles.csv
```

- Negative regexp, same as grep -v

```
awk '!/Article/' articles.csv
```

- Same as grep 'Article' article.csv | grep -v 'Linux'

```
awk '/Article/ && !/Linux/' articles.csv
```

- Lines starting with 1, 2 or 3

```
awk '/^[123]/' articles.csv
```

# Using variables for regexp

- Match if string is found in specific column

```
awk -F, ' $4 ~ "Data" ' articles.csv
```

- Match if date field contains 'Jan' and comments are > 100

```
awk -F, ' $1 ~ /Jan/ && $NF > 100 ' articles.csv
```

- Match a string literally -> "/article/17/1/"

```
awk -F, '/\/article\/17\/1\/' articles.csv
```

```
awk -F, 'index($6, "/article/17/1/")'
articles.csv
```



# WORK

- Which date has the most number of posts?
- How many polls do you find by Jason Baker?
- Calculate the average number of words for Security posts
- How would you calculate the average of words for the 2 first articles of each date?