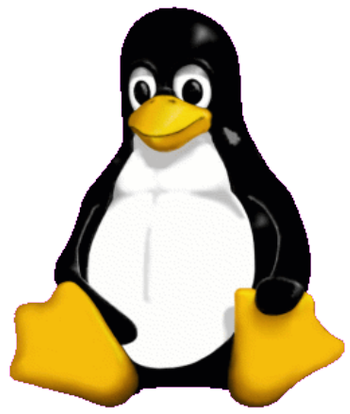
Unix105

BIOL647
Digital Biology

Rodolfo Aramayo



Unix105

Introduction to AWK

Ai Aho, Peter Weinberger, and Brian Kernighan

Unix105

Introduction to AWK

Ai Aho, Peter Weinberger, and Brian Kernighan

C version

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char line[1000], line2[1000];
  char *p;
  double mag;

  while (fgets(line, sizeof(line), stdin) != NULL) {
    strcpy(line2, line);
    p = strtok(line, "\t");
    p = strtok(NULL, "\t");
    p = strtok(NULL, "\t");
    sscanf(p, "%lf", &mag);
    if (mag > 6)    /* $3 > 6 */
      printf("%s", line2);
  }
  return 0;
}
```

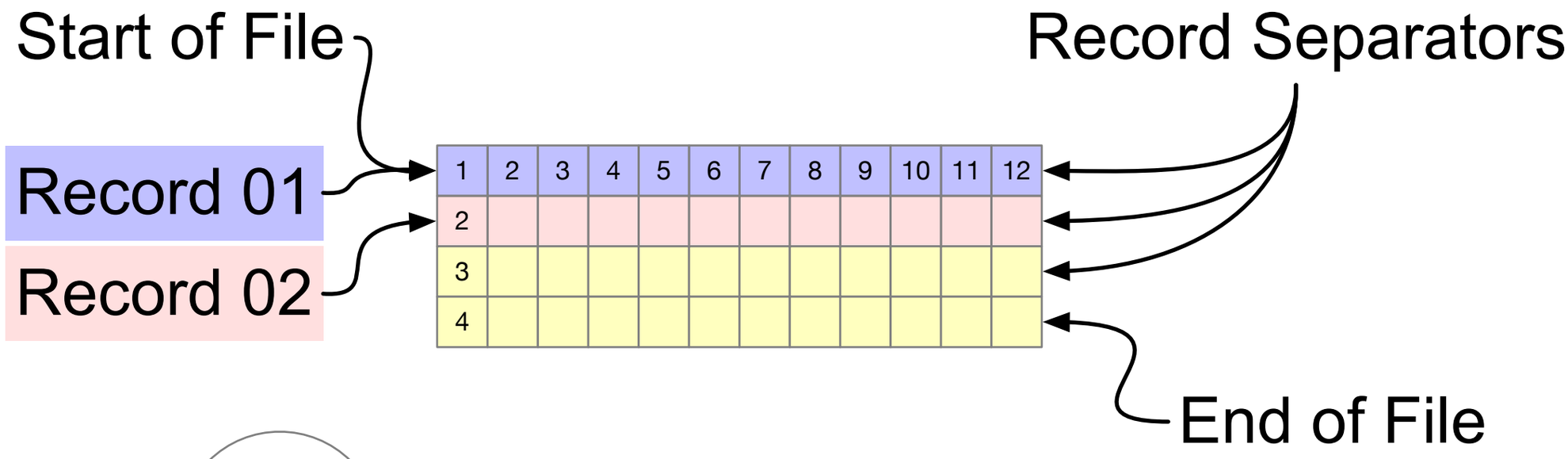
Awk version

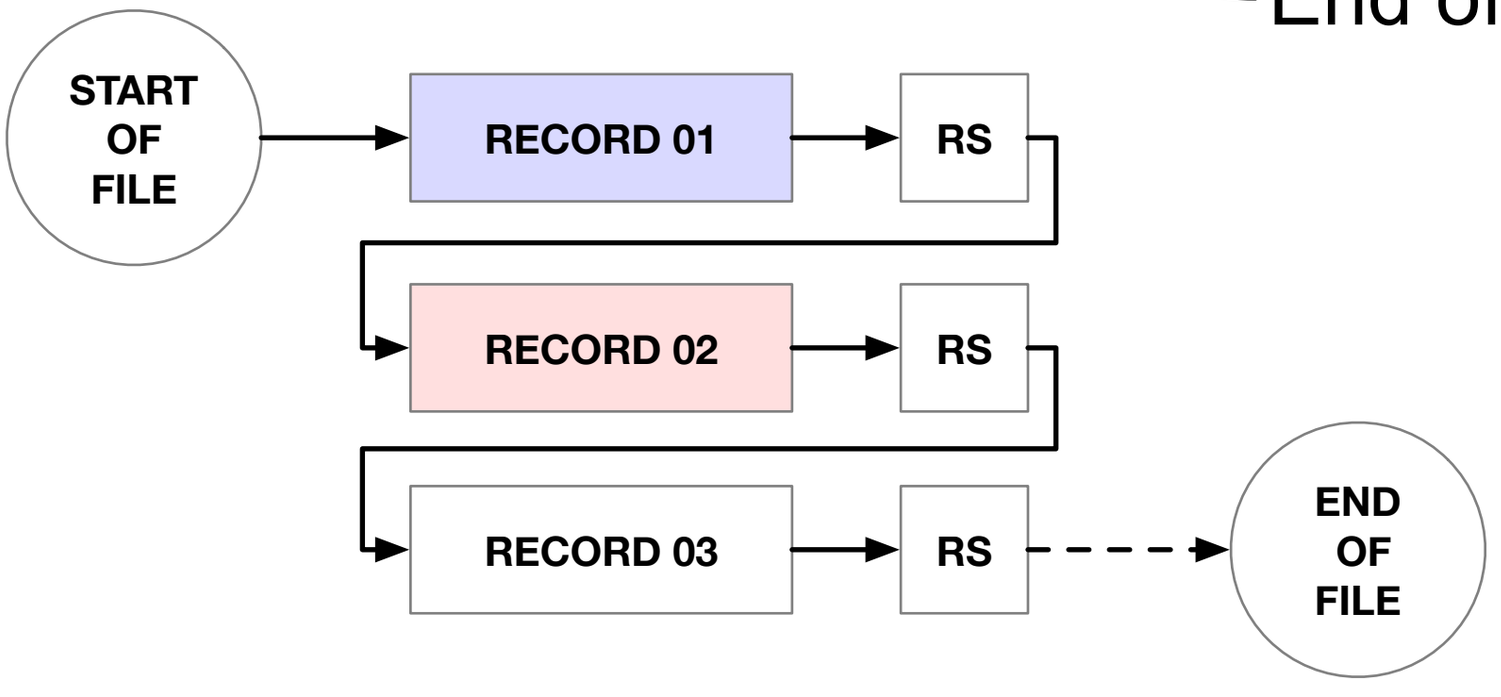
```
$3 > 6
```

Unix105

Introduction to AWK
AWK Records and Fields

- ***Each Record is a line of text (or a row)***
- ***A Record consists of a series of Fields***

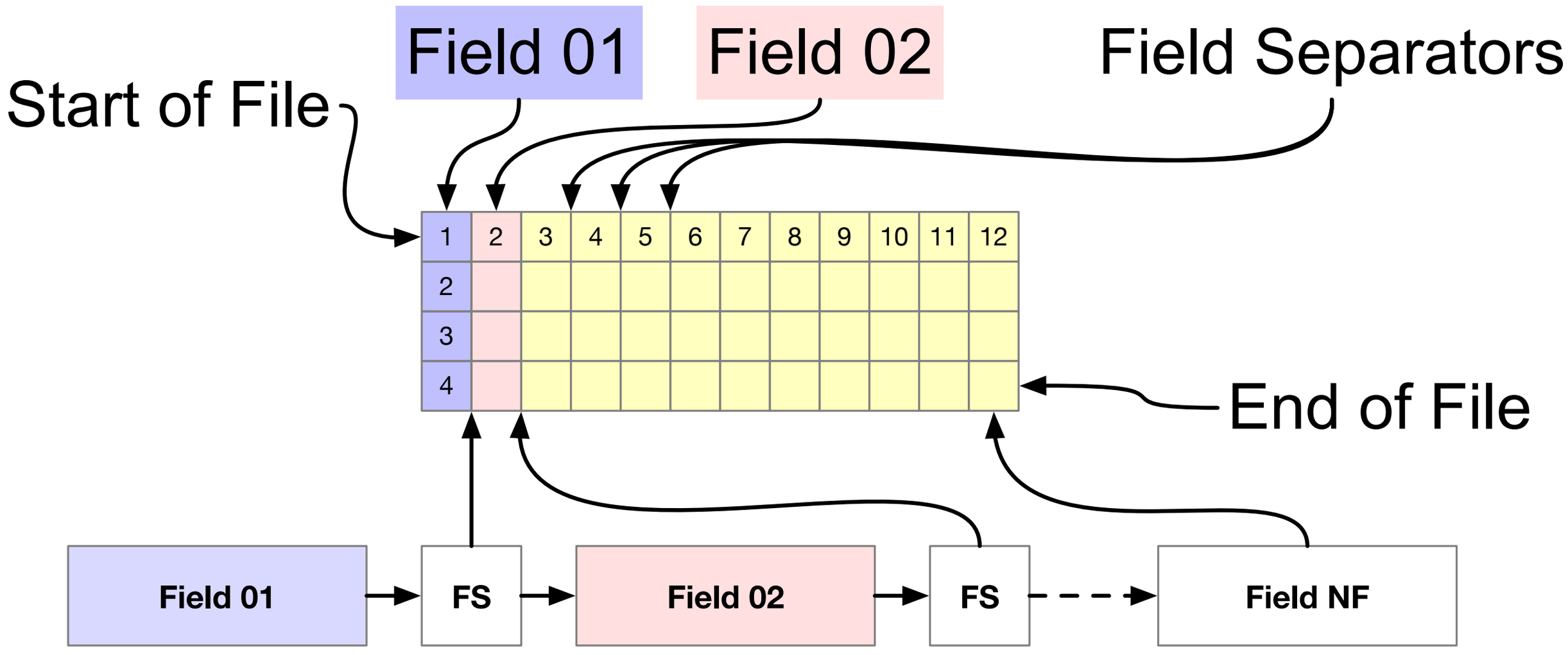




Unix105

Introduction to AWK
AWK Records and Fields

- ***Fields are separated from each other by one or more white spaces, and/or TABs***



Unix105

Introduction to AWK

AWK works best with text files

```
# Let's generate a test file

echo -e "\

Student01Name Student01LastName\n\

Student02Name Student02LastName\n\

Student03Name Student03LastName\n\

Student04Name Student04LastName\n\

Student05Name Student05LastName\n\

Student06Name Student06LastName\n\

Student07Name Student07LastName\n\

Student08Name Student08LastName\n\

Student09Name Student09LastName\

" > names.txt
```

Unix105

Introduction to AWK

AWK works best with text files

- ***awk: Invokes awk***
- ***'awk-program-itself': Single apostrophe, because some of the characters are special to the shell***
- ***{}: Curly Braces tell awk to apply anything inside to the input***
- ***print: Print is a print command***
- ***$2: Specifies the second field on a line***
- ***comma: Specifies that the fields must be separated by a space. This is, it defines a FIELD SEPARATOR (FS)***
- ***$1: Specifies the first field on a line***

```
> cat names.txt
> awk '{print $2,$1}' names.txt
```

Unix105

Introduction to AWK

AWK works best with text files

- ***To print the entire line or Record***

```
> awk '{print $0}' names.txt
> awk '{print}' names.txt
```

Unix105

Introduction to AWK
AWK works best with text files

- ***The command:***

```
> awk '{print $2 $1}' names.txt
```

Results in a field concatenation

Unix105

Introduction to AWK
AWK works best with text files

- ***The command:***

```
> awk '{print $2 ", " $1} names.txt
```

Results in each field being separated by a comma

Unix105

Introduction to AWK
AWK works best with text files

- ***The command:***

```
> awk '{print NF, $0}' names.txt
```

Results in the number of Fields per Record being printed

Unix105

Introduction to AWK

AWK works best with text files

```
# Let’s modify the file to have uneven number of fields:
echo –e "\
Student01Name Student01LastName Observations\n\
Student02Name Student02LastName\n\
Student03Name Student03LastName\n\
Student04Name Student04LastName Observations\n\
Student05Name Student05LastName\n\
Student06Name Student06LastName\n\
Student07Name Student07LastName\n\
Student08Name Student08LastName Observations\n\
Student09Name Student09LastName\
" > names.txt
> cat names.txt # To verify the new file content
```

Unix105

Introduction to AWK
AWK works best with text files

- ***The command:***

```
> awk '{print NF, $0}' names.txt
```

Results in the number of Fields per Record being printed

Unix105

Introduction to AWK

AWK works best with text files

- ***Now, we can select Records that have a given number of fields with the command:***

```
> awk 'NF==3{print NF, $0}' names.txt
```

Unix105

Introduction to AWK
AWK works best with text files

- ***Observe that when you issue the command:***

```
> echo "one two three" | awk '{print $2}'
```

You obtain the desired result because awk interprets the spaces present in the Record as field separators

Unix105

Introduction to AWK
AWK works best with text files

- ***If you were to issue the following command:***

```
> echo "one,two,three" | awk '{print $2}'
```

The command will not work because to awk there is no field 02 in the record

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***To correct the problem we observed before we will define the Field Separator (FS) as follows:***

```
> echo "one,two,three" | awk -F ',' '{print $2}'
> echo "one,two,three" | awk -F , '{print $2}'
> echo "one,two,three" | awk -F, '{print $2}'
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***Note that now the command:***

```
> echo "one two three" | awk –F, '{print $2}'
```

Will not work because awk is looking for commas, not spaces

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***We can also use tabs as Field Separators (FS):***

```
> echo –e "one\ttwo\tthree" | awk –F"\t" '{print $2}'
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***We can also append text, as follows (compare the output of the following commands):***

```
> echo "hello hi" | awk -v hi=HELLO '{print $1,hi}'
> echo "hello awk" | awk -v hi=HELLO '{print $1,hi}'
> echo "hello" | awk -v hi=HELLO '{print $1,hi}'
> echo "nice to meet you" | awk -v hi=HELLO '{print $1,hi}'
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***When you define a Field Separator (FS), AWK stops recognizing the 'default' Field Separators***
- ***Compare the following commands:***

```
> echo "one two,three four five,six" | awk -F ',' '{print $2}'
> echo -e "one\ttwo,three\tfour\tfive,six" | awk -F ',' '{print $2}'
# Output is empty because field 2 is empty
> echo -e "one,,two" | awk -F ',' '{print $2}'
> echo -e ",two,three" | awk -F ',' '{print $2}'
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***You can define your own Field Separator (FS):***

```
> echo "oneABCtwoABCthree" | awk –F 'ABC' '{print $2}'
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***And, you can define RegExp as Field Separators (FS):***

```
> echo "one two three" | awk '{print $2}'
> echo "one,two,three" | awk '{print $2}'  # Field $2 does not exist
> echo "one,two,three" | awk '{print $1}'  # Field $1 does exist
# Defining Field
> echo "one\!two,three" | awk -F '[,!]' '{print $2}'
> echo "one,two,three" | awk -F '[,!]' '{print $2}'
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***or, to bypass Bash***

```
> awk -F '[,!]' '{print $2}'
> one!two,three #<==Enter this
```

Unix105

Introduction to AWK
Defining a Field Separator (FS)

- ***Field Separators (FS) can be specified within an AWK program:***

```
> awk 'BEGIN{FS=","} {print $2}' or awk 'BEGIN{FS=",";};{print $2}'
> "one,two,three" # <==Enter this
> "four,five,six" # <==Enter this
```

Unix105

Introduction to AWK

Defining a Record Separator (RS)

```
> echo -e "\

Student01Name\nStudent01Address\nStudent01City, Student01State Student01ZIPNumber\n\n\

Student02Name\nStudent02Address\nStudent02City, Student02State Student02ZIPNumber\n\n\

Student03Name\nStudent03Address\nStudent03City, Student03State Student03ZIPNumber\n\n\

Student04Name\nStudent04Address\nStudent04City, Student04State Student04ZIPNumber\n\n\

Student05Name\nStudent05Address\nStudent05City, Student05State Student05ZIPNumber\n\n\

Student06Name\nStudent06Address\nStudent06City, Student06State Student06ZIPNumber\

" > StudentsData.txt


> cat StudentsData.txt


> cat StudentsData.txt | \

awk 'BEGIN{RS="";FS="\n"}{name=$1;address=$2;citystatezip=$3;print name "," address "," citystatezip}'
```

Unix105

Introduction to AWK
Manipulating Files

- ***Double spacing a file:***

```
> ln -s /vol_b/zzStorage/Hsapiens_Minimal.Bed .

> cat Hsapiens_Minimal.Bed

> awk '1;{print ""}' Hsapiens_Minimal.Bed | less -S

# ORS – Output Record Separator variable
> awk 'BEGIN{ORS="\n\n"};1' Hsapiens_Minimal.Bed | less -S
```

Unix105

Introduction to AWK
Manipulating Files

- ***Double-space a file so that no more than one blank line appears between lines of text***

```
> awk 'NF { print $0 "\n" }'
```

Where NF = Number of Fields

"If the line is not empty, print the whole line followed by newline."

Unix105

Introduction to AWK
Manipulating Files

- ***Triple spacing a file:***

```
> awk '1;{print "\n"}' Hsapiens_Minimal.Bed | less -S

> awk '1;{print "\n"}' Hsapiens_Minimal.Bed > Hsapiens_Minimal_01.Bed
```

Unix105

Introduction to AWK
Manipulating Files

- ***Precede each line by its line number FOR THAT FILE (left alignment). Using a tab (\t) instead of space will preserve margins***

```
> awk '{print FNR "\t" $0}' Hsapiens_Minimal_01.Bed | less -S
```

Unix105

Introduction to AWK
Manipulating Files

- ***Precede each line by its line number FOR ALL FILES TOGETHER, with tab***

```
> awk '{print NR "\t" $0}' Hsapiens_Minimal*.Bed | less -S
```

Unix105

Introduction to AWK
Manipulating Files

- ***Count lines (emulates “wc -l”):***

```
> awk 'END{print NR}' Hsapiens_Minimal_01.Bed
```

Unix105

Introduction to AWK
Manipulating Files

- ***Delete ALL blank lines from a file (same as "grep '.' "):***

```
> awk NF Hsapiens_Minimal.Bed | less -S

> awk NF Hsapiens_Minimal_01.Bed | less -S

> awk '/./' Hsapiens_Minimal.Bed | less -S

> awk '/./' Hsapiens_Minimal_01.Bed | less -S
```

Unix105

Introduction to AWK
Manipulating Files

- ***Print first 10 lines of file (emulates behavior of "head"):***

```
> awk 'NR < 11' Hsapiens_Minimal.Bed | less -S
```

- ***Print first line of file (emulates "head -1"):***

```
> awk 'NR>1{exit};1' Hsapiens_Minimal_01.Bed | less -S
```

- ***Print the last line of a file (emulates "tail -1"):***

```
> awk 'END{print}' Hsapiens_Minimal_01.Bed | less -S
```

Unix105

Introduction to AWK
Manipulating Files

- ***Print only lines which match regular expression (emulates “grep”):***

```
> awk '/ENSG00000227232/' Hsapiens_Minimal_01.Bed | less -S
```

- ***Print only lines which do NOT match regex (emulates “grep -v”):***

```
> awk '!/ENSG00000227232/' Hsapiens_Minimal_01.Bed | less -S
```

Unix105

Introduction to AWK
Manipulating Files

- ***Delete leading whitespace (spaces and tabs) from the beginning of each line :***

```
> echo -e "     Test"
> echo -e "     Test" | awk '{ sub(/^[ \t]+/, ""); print }'

> echo -e "\t\t\t\tTest"
> echo -e "\t\t\t\tTest" | awk '{ sub(/^[ \t]+/, ""); print }'
```

Unix105

Introduction to AWK
Manipulating Files

- ***Delete trailing whitespace (spaces and tabs) from the end of each line:***

```
> echo -e "Test    " | cat -te
> echo -e "Test    " | awk '{ sub(/[ \t]+$/, ""); print }' | cat -te


> echo -e "Test\t\t\t\t" | cat -te
> echo -e "Test\t\t\t\t"|awk '{ sub(/[ \t]+$/, ""); print }'|cat -te
```

Unix105

Introduction to AWK
Manipulating Files

- ***Delete both leading and trailing whitespaces from each line:***

```
> echo -e "\tTest\t" | cat -te

> echo -e "\tTest\t" | awk '{ gsub(/^[ \t]+|[ \t]+$/, ""); print }'
```

Unix105

Introduction to AWK
Manipulating Files

- ***Substitute (find and replace) from each line:***

```
# Remember?
> echo "upstream and upward" | sed 's/up/down/'
> echo "upstream and upward" | sed 's/up/down/g'

> echo "upstream and upward" | awk '{ sub(/up/,"down"); print }'
> echo "upstream and upward" | awk '{ gsub(/up/,"down"); print }'
```

Unix105

Introduction to AWK
Manipulating Files

- *gensub (regex, s, h[, t])*
  - *It searches the string t for regex and replaces h-th match with s. If t is not given, $0 is assumed. Unlike sub and gsub it returns the modified string t (remember that sub and gsub modified the string in-place)*

```
> echo "upstream and up, up, and upward"|awk '{ sub(/up/,"down"); print }'

> echo "upstream and up, up, and upward"|awk '{ gsub(/up/,"down"); print }'

> echo "upstream and up, up, and upward"|awk '{ $0 = gensub(/up/,"down",3); print }'
```

Unix105

BIOL647
Digital Biology

Rodolfo Aramayo