Edytor strumieniowy `sed` oferuje całkiem sporo możliwości w zakresie obróbki tekstu: podstawianie, dodawanie, usuwanie, zamianę tekstu i oczywiście wyrażenia regularne. Bywają jednak sytuacje, gdy potrzebujemy trochę więcej: zmiennych, tablic, operacji arytmetycznych, instrukcji warunkowych i pętli, czyli typowych elementów programowania. Takie funkcjonalności daje program `awk`. Jego nazwa pochodzi od nazwisk twórców: Aho, Weinberger, Kernighan.

Znane są trzy odmiany tego programu:

- `awk` - stara, oryginalna wersja opracowana przez AT&T,
- `nawk` - nowsza, usprawniona wersja,
- `gawk` - wersja GNU.

Na nasze, ograniczone potrzeby, możemy całkowicie pominąć tę kwestię. Znowu, jak w przypadku sed, programowi `awk` można by poświęcić cały semestr zajęć, a my mamy na to tylko część jednego wykładu.

Program `awk` jest edytorem strumieniowym jak `sed`, tylko ma więcej możliwości. Czyta wiersz po wierszu wskazany plik lub standardowe wejście, dla każdego wiersza osobno wykonuje polecenia ze skryptu i wypisuje wyniki na standardowe wyjście. Większe skrypty zapisujemy w osobnych plikach, natomiast proste, krótkie skrypty można zamieścić w linii poleceń - podobnie z resztą jak w `sed`.

Program `awk` ma tylko dwie istotne opcje:

`-f` : czyta skrypt z podanego pliku zamiast z linii poleceń,
`-Fc` : używa znaku `c` jako separatora pól w plikach wejściowych.

Jego możliwości tkwią w związanym z nim języku programowania. Bez wahania można użyć tego określenia bo tak jak pisaliśmy ma on wszelkie cechy prawdziwego języka programowania.

Skrypty `awk` przypominają trochę programy w C i skrypty shellowe. Ze względu na swoje przeznaczenie do obróbki tekstów skrypt `awk` zawiera jednak specyficzne konstrukcje. Skrypt `awk` podzielony jest na bloki postaci:

```
wzorzec { akcja }
```

Wzorce są bardzo podobne jak w sed. Są to wyrażenia regularne ujęte w ograniczniki `//`. W miejscu wzorca można użyć zakresu utworzonego z dwóch wzorców rozdzielonych przecinkiem. Tutaj w `awk` dodatkowo we wzorcu można używać instrukcji, tak aby powstało wyrażenie logiczne zwracające true lub false. Jeśli przetwarzany wiersz pasuje do wzorca albo wyrażenie logiczne zwraca true to wykonywane są instrukcje akcji. Są dwa specyficzne wzorce: `BEGIN`, `END`. Akcje związane z `BEGIN`, jak łatwo się domyśleć, wykonywane są raz na samym początku, zanim zostanie przetworzony pierwszy wiersz pliku wejściowego. Natomiast akcje związane ze wzorcem `END` wykonane są na zakończenie, po przetworzeniu ostatniego wiersza pliku wejściowego.

Generalnie `awk` został zaprojektowany do przetwarzania plików tekstowych w postaci tabularycznej, gdzie mamy zgromadzone przeróżne dane w formie kolumn. Domyślnie spacja i tabulacja są separatorami kolumn. Można to zmienić za pomocą opcji `-F` albo przypisać odpowiednią wartość specjalnej zmiennej `FS`. Zmienne `$1`, `$2`... zawierają dane z poszczególnych kolumn. Zmienna `$0` zawiera cały przeczytany wiersz. Tak więc:

```
sirius$ echo ”hello world” | awk '{ print $2, $1 }'
world hello
```

Tutaj akurat wzorzec jest pusty, więc pasuje do każdego wiersza. Jak się łatwo domyśleć instrukcja `print` wypisuje dane.

W pliku `/etc/passwd` mamy dane użytkowników systemu. Znakiem dwukropka `:` rozdzielone są tam poszczególne dane. Jeśli interesuje nas tylko nazwa i UID użytkownika to możemy je łatwo odfiltrować:

```
awk -F: '{ print $1"\t"$3 }' /etc/passwd
```

Musimy jednak poinformować program `awk`, że tutaj kolumny rozdzielone są dwukropkiem, nie spacją. Znak `\t` to tabulacja. To samo uzyskamy podstawiając dwukropek do zmiennej `FS`. Musimy to jednak zrobić na samym początku, zanim cokolwiek `awk` przeczyta z pliku wejściowego:

```
awk 'BEGIN {FS=":"} { print $1"\t"$3 }' /etc/passwd
```

Jeśli interesują nas tylko użytkownicy, których nazwa zaczyna się na n, r, lub s to:

```
awk -F: '/^[nrs]/ { print $1"\t"$3 }' /etc/passwd
```

To samo można uzyskać sprawdzając, czy pierwsza kolumna pasuje do naszego wyrażenia regularnego:

```
awk -F: '$1 ~ /^[nrs]/ { print $1"\t"$3 }' /etc/passwd
```

To co do tej pory uzyskaliśmy można uzyskać bez `awk` jakimiś sztuczkami w `sed`, ale znacznie komplikując polecenia. Program `awk` znacznie jednak ułatwia obróbkę plików tabularycznych. Spróbujmy teraz wypisać tylko dane tych użytkowników, których UID jest mniejszy niż 100:

```
awk -F: '$3 < 100 { print $1"\t"$3 }' /etc/passwd
```

To bardzo trudne do uzyskania bez `awk`. Zamiast wzorca logicznego można użyć instrukcji warunkowej `if`. Należy tylko pamiętać o dodatkowych nawiasach klamrowych `{}`:

```
awk -F: '{ if ($3 < 100) { print $1"\t"$3 } }' /etc/passwd
```

Aby zobaczyć jak używa się tablic i pętli w `awk` wypiszmy to samo, ale od końca:

```
awk -F: '{ a[i++]=$1”\t”$3 } \
END {for (j=i-1; j>=0; j--) print a[j] }' /etc/passwd
```

Dla czytelności rozbiliśmy polecenie na dwie linie. W tym celu na końcu pierwszej jest backslash `\` do ochrony znaku nowej linii. Interpretowany jest on przez shella, nie przez `awk`. Równie dobrze moglibyśmy wpisać to w jednej, dłuższej linii:

```
awk -F: '{ a[i++]=$1”\t”$3 } END {for (j=i-1; j>=0; j--) print a[j] }' /etc/passwd
```

W tablicy `a` zapisujemy pola z pierwszej i trzeciej kolumny rozdzielone tabulacją. Robione to jest dla każdego wiersza pliku wejściowego. Zmienna `i` jest licznikiem. Nie musimy jej deklarować przed użyciem. Przy pierwszym użyciu jest tam 0. W bloku END w pętli wypisujemy zawartość tablicy `a` od końca. Zmienna `j` to pomocniczy licznik. Składnia pętli `for` jest taka jak w C.

Istotne zmienne `awk`:

- `FS` - field sparator, separtor pól
- `NR` - number of a row, numer przetwarzanego wiersza
- `NF` - number of fields, ilość kolumn w przetwarzanym wierszu

Przydatne funkcje `awk`:

- `print` - wypisuje podane dane na standardowy wyjście
- `printf` - wypisuje sformatowane dane na standardowy wyjście, składnia jak w C
- `length` - długość napisu
- `split` - rozbija napis na elementy tablicy według określonego znaku
- `match` - dopasowuje wyrażenie regularne do napisu i zwraca true lub false
- `sub` - podstawia ciąg znaków w miejsce dopasowanego wyrażenia regularnego jak w sed
- `gsub` -jak sub tylko z opcją g - global
- `substr` - wybiera podciąg z podanego ciągu znaków
- `tolower` - przekształca wielkie litery na małe
- `toupper` - przekształca małe litery na wielkie

To tylko wierzchołek góry lodowej, bo `awk` posiada znacznie więcej wbudowanych funkcji, także trygonometryczne.