Do tej pory pisaliśmy pojedyncze polecenia, sporadycznie były to 2-3 polecenia rozdzielone średnikiem w jednej linii, albo wywoływane kolejno w osobnych liniach. Były to proste zadania, chociaż dzięki potokom w Unixie wiele takim pojedynczym poleceniem można zdziałać. Niektóre zadania, które spokojnie można wykonać w shellu, wymagają jednak zastosowania większej liczby poleceń. Poza tym przydają się zmienne, instrukcje warunkowe i pętle. One także dostępne są bezpośrednio w shellu, ale lepiej zapisać sobie wykonywane instrukcje w pliku. Wtedy, możemy je modyfikować, poprawiać i używać wielokrotnie.

Powiedzmy, że mamy 300 zdjęć w formacie JPEG zapisane w plikach z rozszerzeniem `.jpeg`. Ponieważ przyzwyczajeni raczej jesteśmy do nadawania takim plikom rozszerzenia `.jpg` chcemy zmienić ich nazwy. Można to zrobić ręcznie, ale po 5-10 plikach stracimy raczej zapał. Do czego mamy shella? Można to spokojnie zrobić tak zwanym *jednolinijkowcem*, to znaczy poleceniem, które wpiszemy bezpośrednio do shella w jednej linii:

```
for f in *.jpeg; do mv $f `echo $f | sed s/\.jpeg$/.jpg/`; done
```

Mamy tutaj wprawdzie dwa średniki, a średniki w shellu rozdzielają kolejne instrukcje. Tak więc nasz jednolinijkowiec to w rzeczywistości 3 instrukcje. Przy okazji przypomnijmy, że *lewe* apostrofy to *command substitution*, czyli instrukcje w nich umieszczone wykonywane są wcześniej, a instrukcja zewnętrzna widzi w tym miejscu wynik. Tutaj przy pomocy `sed` zmieniamy w nazwie pliku rozszerzenie z `.jpeg` na `.jpg`. Instrukcja zewnętrzna, w tym wypadku `mv`, jako pierwszy argument ma oryginalną nazwę w zmiennej `$f` , a jako drugi argument nazwę zmienioną. Skąd sed czyta oryginalną nazwę pliku? Przed `sed` jest instrukcja `echo $f`, która wypisuje na standardowe wyjście tę nazwę. W potoku jest ona przekazywana na standardowe wejście do `sed`. To co `sed` wypisze na standardowe wyjście, czyli zmienioną nazwę, instrukcja `mv` widzi jako swój drugi argument.

Program `sed` jest niezastąpiony, gdy chcemy wykonać poważniejsze zmiany w tekście. Tym tekstem była teraz dla nas nawa pliku. Jak zwykle jednak, to samo zadanie można wykonać inaczej, może nawet prościej jeśli skorzystamy z programu `basename`, który odcina ścieżkę i podane rozszerzenie:

```
for f in *.jpeg; do mv $f `basename $f .jpeg`.jpg; done
```

Jeśli taką operację zmiany nazw plików wykonujemy co jakiś czas, wówczas warto ją uwiecznić w pliku, tak aby nie pisać jej od nowa za każdym razem, narażając się na popełnienie błędu i utratę plików. Do tego celu możemy użyć dowolnego, prostego edytora tekstu jak okienkowy `gedit`, albo konsolowy `nano` albo `pico`. Dla hardcorowców jest `emacs` i `xemacs`, ale ich użytkownicy nie muszą czytać o pisaniu skryptów. Nazwę dla pliku możemy nadać dowolną. Niektórzy dopisują rozszerzenie `.sh` do skryptów shellowych, ale nie jest ani obowiązkowe, ani powszechne.

Dobrym zwyczajem jest za to umieszczanie na początku pliku ścieżki do interpretera shellowego jakiego używamy do uruchamiania naszego skryptu. Jest to ogólnie przyjęty standard specjalnego komentarza, bo komentarze w shellu zaczynają się znakiem `#`, kończą znakiem końca linii i mogą być umieszczone w dowolnym miejscu skryptu. Ten specjalny komentarz musi być w pierwszej linii skryptu i wygląda na przykład tak:

```
#!/bin/sh
```

To dla shella, z którego wykonujemy skrypt, oznacza, że należy uruchomić program `/bin/sh` i do niego przekazać wykonanie instrukcji w tym pliku. Jeśli nasz skrypt zawiera rozszerzenia typowe dla `bash` wtedy wpiszemy:

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

To nie musi być shell. Skrypty pisane w `awk` mogłyby zaczynać się tak:

```
#!/usr/bin/awk
```

Kolejna rzecz, jeśli chcemy uruchamiać skrypt wpisując jego nazwę, to prawa dostępu. Plik wykonywalny jak pamiętamy ma ustawiony flagę `x` (execute), czyli pierwszy bit w uprawnieniach. Przypomnijmy, że

```
chmod +x moj_skrypt
```

doda możliwość wykonywania, właścicielowi, grupie i pozostałym. Teraz można spróbować wywołać napisany skrypt. Jeśli w linii poleceń wpiszemy nazwę to na 99% procent dostaniemy komunikat *command not found*. Dlaczego? Aby uruchomić jakikolwiek program, czy skrypt w Unixie, musi się on znajdować w jednym z miejsc określonych w zmiennej `$PATH`. Aby zobaczyć co tam jest wpiszmy:

```
echo $PATH
```

Tak przy okazji `$PATH` to przykład zmiennej shellowej. Ta ma specjalne znaczenie dla systemu. Problem z `$PATH` można obejść podając pełną ścieżkę do skryptu przy uruchamianiu. Tutaj przydaje się specjalny katalog o nazwie `.` kropka, oznaczający bieżący katalog, aby nie wpisywać pełnej ścieżki absolutnej (bezwzględnej). Tak więc:

```
./moj_skrypt
```

spowoduje wykonanie instrukcji zapisanych w pliku `moj_skrypt`.

Możemy nie zmieniać uprawnień i nie dodawać ścieżki interpretera na początku, ale wtedy musimy ten interpreter wykonać explicite:

```
sh moj_skrypt
```

albo

```
bash moj_skrypt
```

Zadziała to również, gdy już zmieniliśmy uprawnienia i wpisaliśmy ścieżkę interpretera.

Tak więc ostatecznie nasz skrypt mógłby wyglądać tak:

```
#!/bin/sh

for stara_nazwa in *.jpeg
do
    nowa_nazwa=`echo $stara_nazwa | sed s/\.jpeg$/.jpg/`
    mv $stara_nazwa $nowa_nazwa
done
```

Dla czytelności użyliśmy dłuższych, bardziej wymownych, nazw zmiennych oraz umieściliśmy instrukcje do wykonania w osobnych liniach. Człowiek zwyczajowo czyta z góry na dół, nie wszerz. Dlatego tym lepiej im linie są krótsze.