Środowisko pracy informatyka

Laboratorium 8 – przekierowania, potoki i filtry

Spis treści

1 Przygotowanie do pracy w laboratorium

1. Wykonaj polecenie `rm -rf spi-*`
2. Sklonuj swoje repozytorium z `Bitbucketa`.
3. Przejdź do katalogu dla swojego repozytorium:

   `cd spi-<TAB>`

4. Wprowadź niezbędne ustawienia konfiguracji:

   ```
   git config user.name "Imię Nazwisko"
   ```

   ```
   git config user.email "adres-email"
   ```

5. Utwórz plik `.keep` poleceniem `touch`. Dodaj do repozytorium utworzony plik, zatwierdź zmiany i wyślij je na `Bitbucketa`. Utwórz katalog o nazwie `lab08` i przejdź do utworzonego katalogu.

6. W dalszym ciągu, po utworzeniu każdego kolejnego pliku wykonaj poniższe czynności:

   (nazwa `foo.txt` będzie się zmieniać w zależności od polecenia)

   Uwaga 1: Teraz nie wykonuj poniższych poleceń `gita`.

   Uwaga 2: Nazwa `foo.txt` będzie się zmieniać w zależności od polecenia.

   - dodaj plik do śledzenia

     ```
     git add foo.txt
     ```

   - zatwierdź zmiany

     ```
     git commit -m "Dodany plik foo.txt"
     ```

     wyślij zmiany do zdalnego repozytorium

     ```
     git push
     ```

2 Przekierowania

Przekierowania wejścia-wyjścia (ang. i/o redirection). umożliwiają przekierowanie strumienia wejścia oraz wyjścia poleceń do oraz z plików, a także pozwalają na połączenie wielu poleceń w tak zwane potoki.

Aby zademonstrować możliwości tej funkcji, wykorzystane będą następujące polecenia:
`cat` – łączy pliki,
`sort` – sortuje wiersze tekstu,
`uniq` – zgłasza lub pomija powtarzające się wiersze,
`wc` – wypisuje liczbę wierszy, wyrazów oraz bajtów w każdym pliku,
`grep` – wypisuje wiersze pasujące do wzorca,
`head` – zwraca początek pliku,
`tail` – zwraca koniec pliku,
`tee` – pobiera dane ze standardowego wejścia, zapisuje je do plików i wypisuje na standardowym wyjściu.

2.1 Standardowy strumień wejścia, wyjścia oraz błędów

Wiele programów, z których dotychczas korzystaliśmy, zwraca jakieś dane wynikowe. Dane te obejmują zwykle dwa rodzaje informacji. Pierwszy rodzaj to wyniki działania programu; czyli dane, do których wytworzenia przeznaczony jest program. Drugi rodzaj to informacje

o statusie oraz o błędach, czyli informacje o przebiegu działania programu. Weźmy na przykład polecenie ls, którego wykonanie spowoduje wyświetlenie na ekranie wyniku jego działania oraz komunikatów o błędach.

Pamiętając o zasadzie obowiązującej w systemie Unix, głoszącej, że „wszystko jest plikiem", programy takie jak ls tak naprawdę wysyłają wyniki działania do specjalnego pliku zwanego standardowym strumieniem wyjścia (inaczej `stdout`). Natomiast komunikaty o statusie są przekazywane do innego pliku, zwanego standardowym strumieniem błędów (`stderr`). Domyślnie zarówno standardowy strumień wyjścia, jak i standardowy strumień błędów są wyświetlane na ekranie i nie są zapisywane w pliku na dysku.

Ponadto wiele programów pobiera dane z tzw. standardowego strumienia wejścia (`stdin`), który jest domyślnie powiązany z klawiaturą. Przekierowania wejścia-wyjścia pozwalają na zmianę miejsca, do którego przekazywany jest wynik oraz z którego są pobierane dane. W normalnej sytuacji strumień wyjścia jest wypisywany na ekranie, a strumień wejścia jest pobierany z klawiatury, jednak korzystając z przekierowań wejścia-wyjścia, możemy to zmienić.

2.2 Przekierowanie standardowego strumienia wyjścia

Przekierowania wejścia-wyjścia pozwalają na zmianę miejsca, do którego przekazywany jest standardowy strumień wyjścia. Aby przekierować standardowy strumień wyjścia do innego pliku, zamiast wyświetlić go na ekranie, należy skorzystać z operatora przekierowania `>`, po którym powinna się pojawić nazwa pliku. Na przykład zamiast wyświetlać na ekranie wynik działania polecenia `ls`, możemy go przekierować do pliku `ls-output.txt`:

```
[bob@lion lab08]$ ls -l /usr/bin > ls-output.txt
[bob@lion lab08]$ ls -l ls-output.txt
```

W powyższym poleceniu tworzymy listę zawartości katalogu `/usr/bin` w długim formacie i przekazujemy wyniki do pliku `ls-output.txt`. Przyjrzyjmy się treści przekierowanego strumienia wyjścia:

```
ls -l ls-output.txt
```

Widzimy, że powstał duży plik tekstowy. Gdy przyjrzymy się treści, korzystając z polecenia `less`, zobaczymy, że plik ls-output.txt rzeczywiście zawiera wyniki naszego polecenia ls:

```
less ls-output.txt
```

Powtórzmy teraz nasz test przekierowania, jednak tym razem zmienimy nazwę katalogu na nazwę katalogu nieistniejącego:

```
[bob@lion lab08]$ ls -l /bin/usr > ls-output.txt
ls: cannot access /bin/usr: No such file or directory
```

Otrzymaliśmy komunikat o błędzie, ponieważ podaliśmy nieistniejący katalog `/bin/usr`. Dlaczego komunikat o błędzie jest wyświetlony na ekranie a nie przekierowany do pliku? Otóż program `ls` nie przekazuje komunikatów o błędach do standardowego strumienia wyjścia, lecz podobnie jak większość dobrze napisanych programów uniksowych, przekazuje je do standardowego strumienia błędów. Ponieważ w poleceniu przekierowaliśmy tylko standardowy strumień wyjścia, a nie standardowy strumień błędów, wiadomość o błędzie została wyświetlona

na ekranie. Wkrótce dowiemy się, jak przekierować standardowy strumień błędów, jednak najpierw zobaczmy, co się stało z naszym plikiem wynikowym:

```
ls -l ls-output.txt
```

Plik ma iteraz zerową długość. Wynika to z tego, że podczas przekierowywania strumienia wyjścia z wykorzystaniem operatora przekierowania `>` plik docelowy jest zawsze całkowicie nadpisywany. Ponieważ nasze polecenie `ls` nie zwróciło żadnego wyniku, tylko komunikat o błędzie, operacja przekierowania zainicjalizowała nadpisywanie pliku, po czym została zatrzymana z powodu wystąpienia błędu. Wynikiem było opróżnienie pliku. Okazuje się, że jeżeli chcemy opróżnić plik (lub utworzyć nowy pusty plik), możemy skorzystać z następującego polecenia:

```
> ls-output.txt
```

Użycie samego operatora przekierowania, bez żadnego poprzedzającego polecenia, spowoduje opróżnienie zawartości istniejącego pliku lub utworzenie nowego pustego pliku. Jak wobec tego można dodać treść przekierowanego strumienia wyjścia do pliku, zamiast go całkowicie nadpisać? W tym celu korzystamy z operatora przekierowania `>>`, jak poniżej:

```
ls -l /usr/bin >> ls-output.txt
```

Wykorzystanie operatora `>>` pozwala na dodanie treści strumienia wyjścia do pliku. Jeżeli plik jeszcze nie istnieje, zostanie utworzony, tak jak w przypadku użycia operatora `>`. Sprawdźmy, jak to działa:

```
[bob@lion lab08]$ ls -l /usr/bin > ls-output.txt
[bob@lion lab08]$ ls -l /usr/bin >> ls-output.txt
[bob@lion lab08]$ ls -l /usr/bin >> ls-output.txt
[bob@lion lab08]$ ls -l ls-output.txt
```

Powtórzyliśmy polecenie trzykrotnie, otrzymując trzykrotnie większy plik wyjściowy.

2.3 Przekierowanie standardowego strumienia błędów

Przekierowanie standardowego strumienia błędów jest trudniejsze, ponieważ w tym przypadku nie można skorzystać ze specjalnego operatora przekierowania. Aby przekierować standardowy strumień błędów, musimy się odwołać do jego deskryptora pliku. Program może przekazać wyniki do dowolnego z kilku ponumerowanych strumieni plików. Dotychczas, mówiąc o pierwszych trzech spośród tych strumieni plików, nazywaliśmy je standardowym strumieniem wejścia, wyjścia oraz błędów. Natomiast powłoka odwołuje się do nich, korzystając odpowiednio z deskryptorów plików `0`, `1` i `2`. W powłoce możemy skorzystać ze specjalnej składni, która służy do przekierowania plików za pośrednictwem numeru deskryptora pliku. Ponieważ standardowemu strumieniowi błędów odpowiada deskryptor pliku `2`, standardowy strumień błędów można przekierować, korzystając z następującego polecenia:

```
ls -l /bin/usr 2> ls-error.txt
```

Numer deskryptora pliku `2` jest umieszczony tuż przed operatorem przekierowania, dzięki czemu standardowy strumień błędów zostanie przekierowany do pliku `ls-error.txt`.

2.4 Przekierowanie standardowego strumienia wyjścia i standardowego strumienia błędów do jednego pliku

W pewnych sytuacjach może się nam przydać przekierowanie całego wyniku polecenia do jednego pliku. W tym celu należy przekierować jednocześnie standardowy strumień wyjścia i standardowy strumień błędów. Możemy tego dokonać na dwa sposoby. Pierwszy, tradycyjny sposób działa tylko w powłoce starego typu:

```
ls -l /bin/usr > ls-output.txt 2>&1
```

Metoda ta polega na wykonaniu podwójnego przekierowania. Najpierw przekierowujemy standardowy strumień wyjścia do pliku `ls-output.txt`, a następnie przekierowujemy deskryptor pliku `2` (standardowy strumień błędów) do deskryptora pliku `1` (standardowego strumienia wyjścia), korzystając z zapisu `2>&1`.

UWAGA Zauważmy, że kolejność przekierowania ma znaczenie. Przekierowywanie standardowego strumienia błędów musi zawsze nastąpić po przekierowaniu standardowego strumienia wyjścia. W przeciwnym razie metoda ta nie zadziała. W powyższym przykładzie fragment

```
> ls-output.txt 2>&1
```

przekierowuje standardowy strumień błędów do pliku `ls-output.txt`, jednak jeżeli zmienimy kolejność na

```
2>&1 > ls-output.txt,
```

standardowy strumień błędów zostanie wyświetlony na ekranie.

Najnowsze wersje powłoki `bash` pozwalają na wykorzystanie drugiej, udoskonalonej metody takiego złożonego przekierowania:

```
ls -l /bin/usr &> ls-output.txt
```

W tym przykładzie korzystamy z jednego operatora `&>` do przekierowania zarówno standardowego strumienia wyjścia, jak i standardowego strumienia błędów do pliku `ls-output.txt`.

2.5 Usuwanie niepotrzebnych danych wynikowych

Niekiedy wyniki działania polecenia nie są nam potrzebne – chcemy się ich po prostu pozbyć. Dotyczy to zwłaszcza komunikatów o błędach i o statusie. System umożliwia nam to poprzez przekierowanie wyniku do specjalnego pliku o nazwie `/dev/null`. W żargonie programistów uniksowych `/dev/null` jest synonimem czarnej dziury (komputerowego śmietnika). Plik ten pobiera dane wejściowe i nie wykonuje na nich żadnych operacji. Aby pozbyć się komunikatów o błędach zwracanych przez polecenie, należy wykonać polecenie następujące:

```
ls -l /bin/usr 2> /dev/null
```

Więcej na temat `/dev/null` można znaleźć pod adresem:

`http://pl.wikipedia.org/wiki//dev/null`.

2.6 Przekierowanie standardowego strumienia wejścia

Dotychczas nie użyliśmy żadnego polecenia korzystającego ze standardowego strumienia wejścia (w rzeczywistości korzystaliśmy, jednak szczegóły ujawnimy nieco później), dlatego teraz poznamy jedno z nich.

`cat` – łączenie plików Polecenie `cat` odczytuje plik lub kilka plików i kopiuje ich zawartość do standardowego strumienia wyjścia. Składnia polecenia wygląda następująco:

```
cat [plik ...]
```

W większości przypadków możemy traktować `cat` jak polecenie analogiczne do polecenia TYPE w systemie DOS. Możemy z niego korzystać do wyświetlenia zawartości plików bez stronicowania. Na przykład polecenie:

```
cat ls-output.txt
```

wyświetli zawartość pliku `ls-output.txt`. Polecenie `cat` jest zwykle używane do wyświetlania zawartości niewielkich plików tekstowych. Ponieważ program `cat` może przyjmować w postaci argumentów kilka nazw plików, polecenie to może posłużyć do łączenia plików. Załóżmy, że pobraliśmy wielki plik, który został podzielony na kilka części i chcemy go na powrót scalić. Jeżeli pliki noszą następujące nazwy: `movie.mpeg.001 movie.mpeg.002 ...movie.mpeg.099` możemy je scalić, korzystając z następującego polecenia:

```
cat movie.mpeg.0* > movie.mpeg
```

Ponieważ wieloznaczniki są zawsze interpretowane w posortowanej kolejności, argumenty zostaną przekazane w kolejności poprawnej.

Wykonajmy teraz polecenie `cat` bez argumentów:

```
cat
```

Polecenie będzie wyglądać jak zawieszone. Tak nam się tylko wydaje, ponieważ polecenie wykonuje dokładnie to, do czego zostało przeznaczone. Jeżeli nie przekażemy do polecenia `cat` żadnego argumentu, będzie ono odczytywać dane ze standardowego strumienia wejścia. Ponieważ standardowy strumień wejścia jest domyślnie powiązany z klawiaturą, polecenie czeka, aż coś wpiszemy. Spróbujmy:

```
cat
Szybki brązowy lis przeskoczył nad leniwym psem
```

Następnie użyjmy kombinacji klawiszy `<Ctrl+D>` (przytrzymajmy klawisz `Ctrl` i naciśnijmy `D`), aby poinformować polecenie `cat` o osiągnięciu końca pliku (`EOF` – od ang. end-of-file) w standardowym wejściu:

```
cat
Szybki brązowy lis przeskoczył nad leniwym psem.
Szybki brązowy lis przeskoczył nad leniwym psem.
```

W przypadku braku argumentów w postaci nazw plików `cat` kopiuje dane ze standardowego strumienia wejścia do standardowego strumienia wyjścia. Dlatego wprowadzona linia została powtórzona. Metodę tę możemy wykorzystać do tworzenia krótkich plików tekstowych. Załóżmy, że chcemy utworzyć plik `lazy_dog.txt` zawierający tekst z naszego przykładu. Możemy wykonać następujące polecenie:

```
cat > lazy_dog.txt
Szybki brązowy lis przeskoczył nad leniwym psem.
```

Wpiszmy polecenie, a następnie tekst, który chcemy umieścić w pliku. Pamiętajmy o zastosowaniu kombinacji `<Ctrl+D>` na zakończenie. Aby sprawdzić wynik naszych działań, możemy skorzystać z polecenia `cat` i ponownie skopiować zawartość pliku do standardowego strumienia wyjścia:

```
cat lazy_dog.txt
Szybki brązowy lis przeskoczył nad leniwym psem.
```

Gdy już wiemy, że oprócz przyjmowania nazw plików w postaci argumentów polecenie `cat` odczytuje także standardowy strumień wejścia, możemy spróbować przekierować standardowy strumień wejścia:

```
cat < lazy_dog.txt
Szybki brązowy lis przeskoczył nad leniwym psem.
```

Korzystając z operatora przekierowania `<`, zmieniamy źródło standardowego strumienia wejścia z klawiatury na plik `lazy_dog.txt`. Widzimy, że wynik jest taki sam jak w przypadku przekazania nazwy pliku w postaci argumentu. Nie jest to zbyt użyteczne w porównaniu z przekazywaniem argumentu z nazwą pliku, jednak pozwala zademonstrować możliwość wykorzystania pliku jako źródła standardowego strumienia wejścia. Inne polecenia lepiej wykorzystują standardowy strumień wejścia, o czym przekonamy się niebawem.

Przejrzyj zawartość podręcznika polecenia `cat`, aby poznać opcje tego programu:

```
man cat
```

Ćwiczenie

1. Wykonaj polecenie `cat --help`.
2. Wykonaj polecenie `cat --help`, a wynik przekieruj do pliku `cat-help.txt`
3. Dodaj utworzony plik, zatwierdź zmiany i wyślij je na `Bitbucketa`.

2.7 Potoki

Możliwość odczytu danych ze standardowego strumienia wejścia przez polecenia i przesłania ich do standardowego strumienia wyjścia jest wykorzystywana przez funkcję powłoki zwaną potokami. Poprzez użycie operatora `|` (kreska pionowa) możemy przekierować standardowy strumień wyjścia jednego polecenia do standardowego strumienia wejścia innego polecenia:

```
polecenie1 | polecenie2
```

Aby to w pełni zademonstrować, musimy skorzystać z kilku poleceń. Wspomniano już, że poznaliśmy jedno polecenie akceptujące dane ze standardowego strumienia wejścia. Jest to i

Zastosujemy polecenie `less`, aby wyświetlić strona po stronie wyniki dowolnego polecenia, które wysyła je do standardowego strumienia wyjścia:

```
ls -l /usr/bin | less
```

Jest to bardzo przydatne. Korzystając z tej techniki, możemy w wygodny sposób sprawdzić wyniki dowolnego polecenia, które przekazuje dane do standardowego strumienia wyjścia.

2.8 Filtry

Potoki są zwykle wykorzystywane do przeprowadzania złożonych operacji na danych. W jednym potoku możemy uwzględnić kilka poleceń. Zwykle polecenia wykorzystywane w ten sposób są nazywane filtrami. Filtry pobierają dane wejściowe, przetwarzają je w jakiś sposób,

a następnie zwracają wynik. Pierwszym filtrem, który zastosujemy, będzie polecenie `sort`. Wyobraźmy sobie, że chcemy utworzyć jedną listę obejmującą wszystkie programy wykonywalne znajdujące się w katalogach `/bin i /usr/bin`. Lista powinna zostać posortowana i wyświetlona na ekranie:

```
ls /bin /usr/bin | sort | less
```

Ponieważ podaliśmy dwa katalogi, wynik działania polecenia `ls` w normalnej sytuacji miałby postać posortowanych list, po jednej dla każdego katalogu. Włączając do potoku polecenie `sort`, zmieniliśmy dane wynikowe, które teraz mają postać jednej posortowanej listy.

2.8.1 `uniq` – zgłaszanie lub pomijanie powtarzających się wierszy

Polecenie `uniq` jest zwykle używane wraz z poleceniem `sort`. Polecenie `uniq` przyjmuje posortowaną listę danych ze standardowego strumienia wejścia lub z jednego pliku, przekazanego w postaci argumentu (warto sprawdzić stronę man dla polecenia uniq), i domyślnie usuwa z listy wszystkie duplikaty. Dlatego aby się upewnić, że nasza lista nie zawiera duplikatów (czyli dowolnych programów o identycznych nazwach znajdujących się w katalogach /bin i /usr/bin), dodamy do naszego potoku polecenie `uniq`:

```
ls /bin /usr/bin | sort | uniq | less
```

W tym przykładzie korzystamy z `uniq` do usunięcia wszelkich duplikatów z wyjścia polecenia `sort`. Jeżeli chcemy zamiast tego wyświetlić listę duplikatów, możemy dodać do polecenia `uniq` opcję `-d`:

```
ls /bin /usr/bin | sort | uniq -d | less
```

2.8.2 `wc` – wypisywanie liczników wierszy, słów oraz bajtów

Polecenie `wc` (ang. word count – liczba słów) służy do wyświetlania liczby wierszy, słów i bajtów zawartych w plikach. Na przykład:

```
wc ls-output.txt
  7423  59306 467748 ls-output.txt
```

W tym przypadku polecenie to wypisze trzy liczby: liczbę wierszy, wyrazów i bajtów zawartych w pliku `ls-output.txt`. Podobnie jak w przypadku poprzednich poleceń, jeżeli uruchomimy to polecenie bez argumentów, będzie ono akceptować dane ze standardowego strumienia wejścia. Opcja `-l` pozwala ograniczyć wynik do licznika samych wierszy. Dodanie tego polecenia do potoku jest wygodną metodą liczenia danych. Aby sprawdzić liczbę elementów znajdujących się na naszej posortowanej liście, możemy wykonać następujące polecenie:

```
ls /bin /usr/bin | sort | uniq | wc -l
7423 ls-output.txt
```

2.8.3 `grep` – wypisywanie wierszy pasujących do wzorca

Polecenie `grep` to potężny program, służący do wyszukiwania wzorców tekstowych w plikach w następujący sposób:

```
grep wzorzec [plik...]
```

Gdy polecenie grep odnajdzie w pliku „wzorzec”, wyświetli wiersze, które ten wzorzec zawierają. Wzorce, które może dopasować polecenie grep, mogą być bardzo skomplikowane, jednak na razie skupimy się na prostym dopasowaniu tekstu.

Załóżmy, że chcemy znaleźć na naszej liście programów wszystkie pliki, których nazwa zawiera fragment zip. W ten sposób możemy się zorientować, które programy w naszym systemie są związane z kompresją plików. W tym celu możemy wykonać następujące polecenie:

```
[bob@lion lab08]$ ls /bin /usr/bin | sort | uniq | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

Do dyspozycji mamy kilka przydatnych opcji: -i, która sprawia, że grep będzie ignorować wielkość liter podczas wyszukiwania (w normalnym przypadku wyszukiwanie jest wrażliwe na wielkość liter), oraz -v, która sprawia, że grep będzie wyświetlać tylko wiersze, które nie pasują do wzorca.

2.8.4 head (tail) – zwracanie początku (końca) pliku

Czasem nie są nam potrzebne wszystkie dane zwracane przez polecenie. Może nam wystarczyć tylko kilka pierwszych lub kilka ostatnich wierszy. Polecenie head wyświetla pierwszych 10 wierszy pliku, a polecenie tail – 10 ostatnich. Domyślnie obydwa polecenia wyświetlają 10 wierszy tekstu, lecz można to zmienić, korzystając z opcji -n:

```
head -n 5 ls-output.txt
head -n5 ls-output.txt
head -5 ls-output.txt
```

Poleceń tych możemy używać także w potokach:

```
ls /usr/bin | tail -n 5
```

Polecenie tail przyjmuje opcję pozwalającą na wyświetlenie zawartości plików w czasie rzeczywistym. Jest to przydatne podczas monitorowania zawartości plików logów w trakcie ich zapisywania. W kolejnym przykładzie przyjrzymy się zawartości pliku messages, który znajduje się w katalogu /var/log. W niektórych dystrybucjach systemu Linux do wykonania tego polecenia będą potrzebne uprawnienia użytkownika uprzywilejowanego, ponieważ plik /var/log/messages może zawierać informacje dotyczące zabezpieczeń:

```
tail -f /var/log/messages
```

Jeżeli skorzystamy z opcji `-f`, polecenie `tail` będzie kontynuować monitorowanie zawartości pliku i wszystkie nowe wiersze będą się natychmiast pojawiać na ekranie. Polecenie to będzie działać, dopóki nie użyjemy kombinacji klawiszy `<Ctrl+C>`.

2.8.5 `tee` – pobieranie danych ze standardowego strumienia wejścia, przekazywanie ich do standardowego strumienia wyjścia i do plików

Program `tee` odczytuje dane ze standardowego strumienia wejścia i kopiuje je zarówno do standardowego strumienia wyjścia (pozwalając danym na dalszy przepływ w potoku), jak i do jednego pliku lub kilku plików. Przydaje się to do przechwytywania danych przepływających przez potok w pośrednim etapie procesowania. Powtórzymy teraz jeden z wcześniejszych przykładów, jednak tym razem uwzględniając polecenie `tee`, którym przechwycimy całą zawartość katalogu do pliku `ls.txt`, zanim polecenie grep przefiltruje zawartość potoku:

```
ls /usr/bin | tee ls.txt | grep zip
```

Mechanizm przekierowań, dostępny w wierszu poleceń, jest niezwykle przydatny w rozwiązywaniu problemów. Wiele poleceń wykorzystuje standardowe strumienie wejścia i wyjścia, a niemal wszystkie programy wiersza poleceń korzystają ze standardowego strumienia błędów, służącego do wyświetlenia komunikatów informacyjnych.

3 Łączenie poleceń – kolejne przykłady

1. Otwórz rozdział spod adresu `https://pdf.helion.pl/wiepol/wiepol.pdf`
2. Pobierz ze strony przedmiotu na portalu `enauka` plik `wplruep.zip` do podkatalogu `lab08`.
3. Rozpakuj pobrany plik poleceniem:

   ```
   unzip wplruep.zip
   ```

4. Wykonaj po kolei wszystkie przykłady (linie zaczynające się od znaku `$`) dla każdego polecenia z sekcji **„Sześć poleceń na dobry początek”**. Po wykonaniu przykładów dla danego polecenia wykonaj je jeszcze raz przekierowując jego wyniki do pliku o nazwie `polecenie-n.txt`, gdzie `n` to numer polecenia z tej sekcji.

4 Zakończenie pracy w laboratorium

1. Przejdź do katalogu nadrzędnego.
2. Sprawdź status repozytorium.
3. Dodaj ewentualne zmiany do repozytorium.
4. Zatwierdź zmiany w repozytorium i wyślij je na `Bitbucketa`.
5. Przejdź do katalogu domowego (`cd`) i usuń katalog repozytorium (`rm -rf spi-*`).