Technologie informacyjne
1 wykład, wprowadzający

Andrzej Giniewicz

28.02.2024

Materiały z pierwszego, wprowadzającego wykładu.

1 Jak dawniej wyglądało korzystanie z komputerów

Aby zrozumieć niektóre terminy związane z obsługą komputera, cofnijmy się do 1910 roku, kiedy wyprodukowano pierwszy dalekopis. Dalekopis miał być urządzeniem, które miało zastąpić telegraf i być dostępne do użytku domowego. Było to proste urządzenie posiadające drukarkę oraz klawiaturę. Naciśnięcie klawisza powodowało zakodowanie litery do sygnału elektrycznego, który był przesyłany przewodem przez centralę do innego odbiorcy. U osoby tej sygnał był odbierany, zamieniany na literę i od razu drukowany na papierze. Dalekopisy nie posiadały pamięci, więc wszystkie znaki do nich przesłane, jeśli nie zostały wydrukowane, ginęły. Do łączenia używano usługi Teleks (ang. Telegraph Exchange), która działała na zasadzie centrali podobnych do centrali telefonii stacjonarnej. W Polsce ostatni operator telekomunikacyjny, który świadczył usługę Teleks, wyłączył ją w 2007 roku, 97 lat po tym, jak do produkcji oddano pierwszy dalekopis.

Dalekopis, Siemens T100, model z biurkiem.
Źródło: `https://www.robkalmeijer.nl/techniek/telecom/telex`.

Dawniej komputery były zbyt duże, aby zmieścić się w pokoju, w którym na nich pracowano. Przykładowo komputer Harvard Mark I z 1939 roku był długi na ponad 15 metrów, miał 2,4 metra wysokości, w środku znajdowało się 805 kilometrów przewodów, a całość ważyła 5,5 tony. Ogólnie to nie przesada, jeśli powimy, że niekiedy komputery zajmowały

całe piętra. Korzystanie z takich komputerów odbywało się poprzez **konsolę**. Na konsoli znajdowały się przełączniki pozwalające wybrać adres pamięci oraz przyciski do umieszczania w nich danych oraz kontrolowania stanu procesora. Na konsoli były również żarówki wyświetlające stan konkretnych adresów pamięci lub procesora.

Konsola komputera PDP-11.
Źródło: `http://www.columbia.edu/cu/computinghistory/pdp11.html`.

Korzystanie z komputera bezpośrednio z konsoli, nie było praktyczne, ale konieczne, aż do lat 50-tych i 60-tych XX wieku, kiedy zaczęto korzystać z łatwo wtedy dostępnych dalekopisów, aby umieścić tak zwany **terminal** w jednym lub większej liczbie biur i połączyć je przewodami z komputerem znajdującym się w innym budynku. Angielski termin *terminal* oznacza coś znajdującego się „na końcu", komputerowy terminal znajdował się na końcu przewodu prowadzącego do komputera.

Dalekopis z modułem do łączenia się z siecią Teleks (moduł do „wybierania numeru" znajduje się po prawej stronie urządzenia) oraz Ken Thompson i Dennis Richie, twórcy języka C oraz systemu Unix, pracujący przy terminalu wykonanym z analogicznego modelu dalekopisu (brak modułu do wybierania numeru). W tle komputer PDP-11 wyposażony w konsolę, tuż za prawym terminalem.
Źródło: `http://www.baudot.net/` oraz `https://computerhistory.org`.

Pod koniec lat 60-tych i w latach 70-tych XX wieku zaczęły pojawiać się urządzenia wzbogacone o niewielką ilość pamięci oraz ekran, które były dedykowane do pracy z komputerami. Pamięć dostępna w takim urządzeniu pozwalała na przechowanie tylu znaków, ile na raz mieściło się na ekranie, dzięki czemu cały czas mogły wyświetlać na ekranie

ostatnio widziane znaki. Terminale takie nazywano terminalami video, ale wciąż nie były to samodzielne komputery — nie mogły działać bez podłączenia do innego komputera.

Terminal Datapoint 3300, jeden z pierwszych terminali wyposażonych w pamięć oraz ekran. Źródło: `https://www.vintagecomputer.net/CTC/3300/`.

W latach 80-tych XX wieku popularność zyskały 8-bitowe mikrokomputery. Ze względu na łatwą dostępność, zaczęły one zastępować fizyczne terminale. Pozwalały one na łączenie się z innym komputerem za pomocą portu RS-232, nazywanych portem szeregowym, oraz specjalnych programów, które pozwalały przesłać na port tekstu i wyświetlić tekst w odpowiedzi. Od tego czasu mianem terminala nazywano program, który służy do połączenia się z innym komputerem.

Interfejs portu szeregowego podłączony do komputera Commodore 64 oraz program CaTer, podłączony jako terminal do innego komputera, na którym uruchomiony jest Linux. Źródło: `https://www.simulant.uk/` oraz `https://sourceforge.net/projects/cater/`.

Ponieważ ludzie byli przyzwyczajeni do uruchamiania programów w ten sposób, chciano również udostępnić możliwość korzystania z programów lokalnie dostępnych na mikrokomputerze w ten sam sposób, co z tych dostępnych zdalnie. W tym celu powstał **pseudo-terminal**. Pseudo-terminal to program, który udaje prawdziwy terminal podłączony do innego komputera, a w rzeczywistości działa na tym samym komputerze, przekazując do innego programu tekst jako wejście oraz odbierając od niego tekst jako wyjście.

Ponieważ wiele terminologii jest dziś mocno „przeterminowana", mówiąc **terminal** będziemy mieli na myśli program, służący do połączenia się z komputerem znajdującym się w innym miejscu lub program służący do połączenia się z pseudo-terminalem znajdującym się na tym samym komputerze. Jeśli będziemy chcieli podkreślić, że mowa o urządzeniu (historycznym) do łączenia się z komputerem, będziemy mówić o **fizycznym terminalu**.

2 Sposoby korzystania z systemów operacyjnych

Z większości funkcjonalności systemu operacyjnego możemy korzystać na dwa sposoby. Pierwszy z nich wykorzystuje graficzny interfejs użytkownika (GUI, ang. Graphical User Interface), drugi linię komend (CLI, ang. Command Line Interface). Gdy korzystamy z jednego z tych interfejsów, mówimy, że wykorzystujemy **powłokę** graficzną (GUI) lub tekstową (CLI).

Podczas gdy powłoki graficzne udostępniają ograniczony interfejs oparty o klikanie, przeciąganie, przesuwanie i okazjonalne wpisanie nazwy pliku, powłoki tekstowe oferują rozbudowane języki programowania, które pozwalają zautomatyzować wiele codziennych czynności. O ile w samej powłoce tekstowej nie obejrzymy zdjęć z wakacji, możemy automatycznie posegregować dziesiątki tysięcy zdjęć do katalogów zależnych od geolokalizacji, czyli tak, aby zdjęcia z wycieczki na Islandię były w odpowiednim katalogu. Za pomocą powłoki tekstowej nie obejrzymy zdjęć w konkretnym ogłoszeniu na portalu z ofertami zakupu mieszkań (choć przeczytamy treść ogłoszenia, są przeglądarki internetowe działające w powłoce tekstowej), ale możemy ściągnąć wszystkie zdjęcia mieszkań dwupoziomowych w Gdańsku kosztujących mniej niż 400 tysięcy złotych. Zasadniczo, jeśli robimy coś, co nie wymaga od nas oglądania zdjęć lub filmów, nie wymaga rozróżniania koloru tekstu lub tła, to powłoka tekstowa daje większe możliwości osobom, które poznały odpowiedni język programowania wykorzystywany przez używaną powłokę. Podczas zajęć będziemy korzystać z powłoki BASH, która dostępna jest na wszystkich trzech popularnych systemach operacyjnych — na Linux oraz macOS dostępne w każdej instalacji[1], natomiast na Windowsie po instalacji dodatkowego oprogramowania, na przykład Git BASH będącego składową instalacji programu Git.

Drugim rozróżnieniem sposobu korzystania z systemu operacyjnego komputera, jest lokalizacja, w której komputer się znajduje. Jeśli fizycznie siedzimy przed komputerem, z którego korzystamy, korzystamy z powłoki lokalnie. Jeśli za pośrednictwem Internetu korzystamy z komputera, znajdującego się w innym miejscu, korzystamy z powłoki zdalnie. Zarazem powłoka tekstowa, jak i powłoka graficzna, pozwala na korzystanie lokalne i zdalne. Korzystanie z powłoki graficznej w sposób zdalny wymaga oprogramowania do zdalnego pulpitu. Jest to rzadko stosowane, ponieważ przesył obrazu wymaga stosunkowo dużo transferu. Dodatkowo zwykle aplikacje do zdalnego pulpitu są jednymi ze słabiej zabezpieczonych usług sieciowych. Niemniej jednak zdarza się, że skorzystanie ze zdalnego pulpitu jest konieczne. W takiej sytuacji warto skorzystać ze standardu dostępnego na wszystkich popularnych systemach, jakim jest VNC (ang. Virutal Network Computing). Alternatywą dla korzystania zdalnie z powłoki graficznej, jest zdalne korzystanie z powłoki tekstowej. Najczęściej jest to jedyna możliwość udostępniana przez serwery oraz superkomputery do obliczeń naukowych. Standardem łączenia się zdalnie z powłoką tekstową, jest tak zwane SSH (ang. Secure Shell). Po wywołaniu odpowiedniej komendy w powłoce tekstowej, wszystkie następujące po niej komendy, będą wykonane na komputerze, z którym się połączyliśmy, a nie na tym, przy którym siedzimy.

[1] System macOS oraz niektóre dystrybucje systemu Linux mogą mieć inną domyślną powłokę, na przykład ZSH, jednak nie przeszkadza to w pisaniu skryptów korzystając ze składni innej powłoki.

W powłoce tekstowej komunikacja odbywa się poprzez wpisywanie odpowiednich komend. Podczas zajęć będziemy korzystać z powłoki BASH, jednak nie jest to jedyna powłoka, jaką możemy spotkać. BASH jest domyślną powłoką na większości dystrybucji systemu Linux, który będzie najczęściej spotykany na serwerach, zatem przy pracy zdalnej, najczęściej będziemy spotykać BASH. Do innych powłok możemy zaliczyć ZSH (domyślną na MacOS, dostępną na Linuxie) lub FISH. Ciekawym nowym projektem powłoki jest NuShell[2], który jest popularny wśród osób zajmujących się analizą danych. Na systemie Windows używana jest powłoka Command Shell (dostępna w komendzie `cmd.exe`) oraz Power Shell.

Co ciekawe, choć korzystanie z powłoki tekstowej kojarzy się głównie z system Linux, system Windows nadgania, wprowadzając w nowych wydaniach technologie dostępne na innych systemach od wielu lat. Od systemu Windows 10 (aktualizacja z jesieni 2018) mamy na systemie Windows dostępny pseudo-terminal, nazywany ConPTY. Od systemu Windows 11 (aktualizacja z jesieni 2022) mamy z kolei nową domyślną aplikację terminala, nazywaną Windows Terminal. Również PowerShell jest dynamicznie rozwijany. Pokazuje to, że jest rzeczywiste zapotrzebowanie na nowoczesną powłokę tekstową również na systemie, który słynie głównie z powłoki graficznej.

3 Anatomia komunikacji przez powłokę tekstową

Kiedy włączamy powłokę tekstową, widzimy tak zwaną zachętę. Istnieje wiele możliwości formatowania zachęty[3], jednak zwykle pojawi się nazwa użytkownika, nazwa komputera oraz obecny katalog roboczy, przykładowo

```
użytkownik@komputer: ścieżka $
```

Symbol `$` nazywamy znakiem zachęty. Jeśli go widzimy, znaczy to, że możemy zacząć wpisywać komendę. Komenda składa się z nazwy oraz potencjalnie opcji oraz argumentów. Opcje i argumenty oddzielamy jednym lub więcej znakami spacji.

Komenda do zmiany ścieżki nazywa się `cd`, natomiast do wypisywania zawartości katalogu `ls` do wypisywania zawartości katalogu oraz `pwd` do sprawdzania, w jakim katalogu jesteśmy. Te trzy komendy posłużą nam jako przykład, więcej będzie okazja poznać na laboratoriach.

W przykładach będziemy zakładać, że użytkownik nazywa się `pi` i korzysta z komputera o nazwie `matplaneta`. Zachęta powinna wyglądać wtedy mniej więcej następująco.

```
pi@matplaneta: ~ $
```

[2] `https://www.nushell.sh/`

[3] Przykładem niestandardowej zachęty jest na przykład `https://starship.rs/`.

Widzimy dzięki temu jako kto i gdzie jesteśmy zalogowaniu, co jest bardzo ważne, jeśli korzystamy ze zdalnego logowania do innych komputerów[4]. Wiemy również, po ~ w katalogu roboczym, że znajdujemy się w katalogu domowym, czyli `/home/pi`, `/Users/pi` lub `C:\Users\Pi`, zależnie od tego, na jakim systemie operacyjnym jesteśmy. Pełną ścieżkę do katalogu roboczego możemy wypisać komendą `pwd`. Po znaku zachęty wpisujemy `pwd` i naciskamy enter.

```
pi@matplaneta: ~ $ pwd
/home/pi
pi@matplaneta: ~ $
```

W nowej linii pojawi się odpowiedź, jaki katalog jest obecnym katalogiem roboczym, po czym pojawi się kolejna zachęta, oznaczająca, że powłoka jest gotowa na przyjęcie kolejnej komendy. Jeśli wpiszemy `ls`, co wypisze katalogi, zobaczymy na przykład

```
pi@matplaneta: ~ $ pwd
/home/pi
pi@matplaneta: ~ $ ls
Desktop             Downloads           ...
Documents           ...                 ...
```

gdzie w miejscu wielokropków pojawi się więcej katalogów oraz plików, znajdujących się w obecnym katalogu roboczym. Instrukcja `ls` posiada wiele opcji, które pozwalają na przykład wypisać elementy w formie listy z dodatkowymi informacji (`-l` jak „list”), wyświetlić również ukryte pliki (`-a` jak „all”) oraz sortować według rozmiaru (`-S` jak „size”). Opcje możemy wypisać jedna po drugiej (`-a -l -S`) lub połączyć je w jeden ciąg (`-alS`).

```
pi@matplaneta: ~ $ ls -alS
-rw-r--r-- 1 pi pi   173412   11-17 11:25  Downlaods
-rw-r--r-- 1 pi pi     3834   11-17 13:25  .config
...
```

W informacjach wypisanych na początku znajduje się informacja o prawach dostępu do plików oraz właścicielu pliku, potem znajduje się rozmiar w bajtach, data i godzina modyfikacji oraz nazwa katalogu. To jak zapisane są prawa dostępu, będzie omówione na dalszych laboratoriach — w tym momencie chcemy zobaczyć, jak do komendy dodać opcje. Częstym błędem jest pominięcie spacji, wtedy otrzymamy komunikat, że nie istnieje komenda `ls-alS`. Innym częstym błędem jest pominięcie symbolu - przed opcjami, wtedy otrzymamy informacje, że nie ma pliku ani katalogu `alS`.

Przykładem komendy, której zwykle używamy z argumentami jest komenda `cd`, która zmienia katalog roboczy.

```
pi@matplaneta: ~ $ pwd
```

[4] `https://www.youtube.com/watch?v=tLdRBsuvVKc` — opis tego, jak pracownik firmy Gitlab mając otwarte dwa terminale prowadzące do różnych komputerów, wykonał komendę nie w tym okienku, w którym chciał i przez przypadek skasował bazę danych.

```
/home/pi
pi@matplaneta: ~ $ cd Downloads
pi@matplaneta: Downloads $ pwd
/home/pi/Downloads
pi@matplaneta: Downloads $
```

Zwróćmy uwagę na zmianę katalogu roboczego z `~` na `Downloads`. Aby przejść do katalogu wyżej, użyjemy `cd ..`, natomiast aby przejść do katalogu domowego nie ważne, w jakim katalogu się znajdujemy, możemy napisać `cd ~` lub `cd`.

```
pi@matplaneta: ~ $ cd Downloads
pi@matplaneta: Downloads $ cd ..
pi@matplaneta: ~ $
```

Aby łatwiej zapamiętać komendy, możemy skojarzyć nazwy komend z angielskimi poleceniami

pwd print working directory,

cd change directory,

ls list directory contents.

Jeśli znamy komendę, ale chcemy dowiedzieć się, jak jej używać, możemy napisać

```
pwd --help
cd --help
ls --help
```

co niezależnie od tego, na jakim systemie uruchomimy BASH, wyświetli pomoc dotyczącą komendy. Możemy też wpisać komendę na stronie `https://explainshell.com/`. Przykładowo

- `https://explainshell.com/explain?cmd=pwd`,
- `https://explainshell.com/explain?cmd=cd%20Downloads`,
- `https://explainshell.com/explain?cmd=ls+-alS`,

objaśni nam działanie przykładów z wykładu, podczas gdy

- `https://explainshell.com/explain/1/pwd`,
- `https://explainshell.com/explain/1/ls`

wyświetlą kompletną dokumentację ze spisem możliwych opcji. W przykładzie nie podano `cd`, ponieważ nie ma dostępnej strony z dokumentacją (`cd --help` zawiera całą dokumentację).

Na koniec zastanówmy się, skąd system bierze komendy. Komendy dzielimy na dwie grupy: programy oraz komendy powłoki. Do programów należą między innymi poznane `ls` oraz `pwd`, podczas gdy `cd` jest komendą powłoki. Oznacza to, że na dysku twardym gdzieś znajdują się programy `ls` oraz `pwd` (w przypadku systemu Windows będzie to `ls.exe` oraz `pwd.exe`), natomiast nigdzie nie ma programu `cd`. Komenda `which` sprawdza, w jakim katalogu znajduje się program.

```
pi@matplaneta: ~ $ which ls
/usr/bin/ls
```

Co jest standardowym miejscem instalacji programu `ls` na systemie Linux. Gdyby `ls` było komendą powłoki, program odpisałby, że nie może znaleźć programu `ls`[5]. Skąd system operacyjny wie, gdzie szukać programów? Nie sprawdza całości dysku twardego, ponieważ nie jest to praktyczne ze względów wydajnościowych. Dodatkowo łatwo byłoby podmienić program na inny, robiący coś nieoczekiwanego. Dlatego systemy operacyjne szukają programów w tak zwanej **ścieżce systemowej**, która jest zapisana w zmiennej środowiskowej `PATH`. Aby sprawdzić, jaka jest ścieżka, możemy skorzystać z komendy `echo`.

```
pi@matplaneta: ~ $ echo $PATH
katalog1:katalog2:katalog3:...
```

Wynikiem działania komendy będą katalogi oddzielone znakiem dwukropka. System operacyjny kolejno sprawdza w ścieżce, od lewej do prawej, czy w danym katalogu nie ma przypadkiem programu o zadanej nazwie. Jeśli znajdzie program o wskazanej nazwie, uruchomi go. Jeśli sprawdzi we wszystkich katalogach i nie znajdzie ani jednej takiej samej nazwy, zgłosi błąd.

Jeśli chcemy uruchomić program, który nie znajduje się w ścieżce, możemy to zrobić na dwa sposoby: dodać nowy katalog do ścieżki systemowej[6] lub użyć pełnej ścieżki w miejscu nazwy komendy. Zakładając, że w katalogu `~/projekty/hello_world` istnieje program o nazwie `program1`, podanie pełnej ścieżki może wyglądać następująco.

```
pi@matplaneta: ~ $ ~/projekty/hello_world/program1
Hello World
```

[5] Sprawdź, czy `which` jest komendą powłoki, czy programem.

[6] To, w jaki sposób edytować zmienne środowisko w tym `PATH`, będziemy przerabiać na dalszych zajęciach.