Narzędzia programistyczne

Przydatne narzędzia

- **kompilator** + edytor tekstu
- **debugger** (odpluskwiacz) - służy do dynamicznej analizy programów (w czasie działania), w celu odnalezienia i identyfikacji zawartych w nich błędów, np. śledzenie wartości zmiennych, stan stosu
- **profilowanie** - badanie zachowania programu, przy użyciu informacji zdobytych podczas jego wykonywania np. graf wywołań, pomiar czasu wykonywania instrukcji, badanie zajętości pamięci
- **analiza statyczna kodu (linter)** - analiza fragmentów kodu (bez konieczności uruchamiania) w calu wykrycia potencjalnych błędów, naruszeń stylu (tzw. *code smells*)
- **systemy kontroli wersji** - śledzenie zmian w kodzie i współdzielenie kodu w zespołach, scalanie zmian
- środowiska IDE - integrują wiele narzędzi

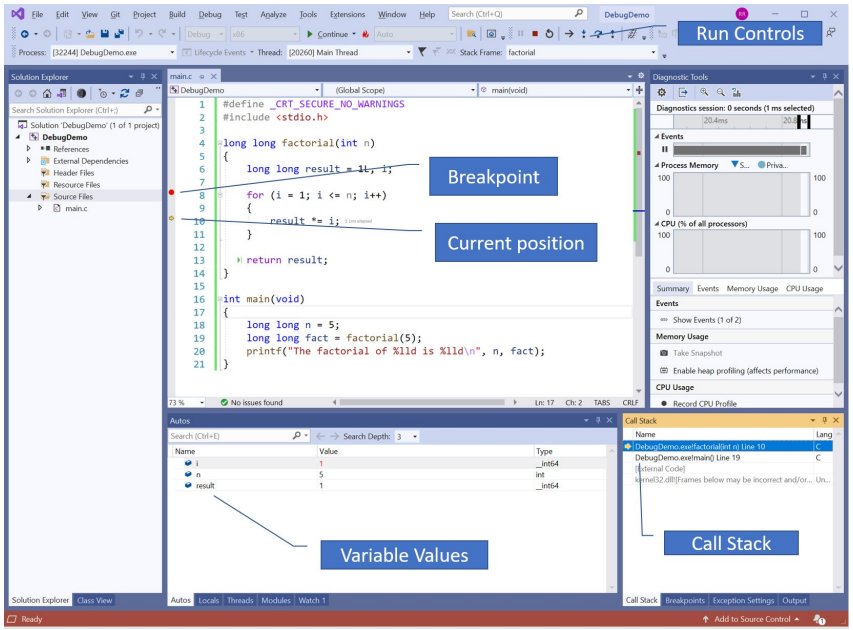
Przydatne opcje kompilatora GCC:

`-Wall`	włacza szereg reguł wykrywających potencjalne błedy, generuje ostrzeżenia
`-Wextra`	włacza dodatkowe reguły dotyczące ostrzeżeń
`-ansi`	używa standardu ANSI C89
`-pedantic`	pilnuje restrykcyjnie norm standardu ISO
`-fanalyzer`	analiza statyczna kodu
`-g`	dodaje do wynikowego kodu informacje niezbędne do debugowania
`-pb`	dodaje do wynikowego kodu informacje niezbędne do profilowania
`-lm`	dołącza bibliotekę matematyczną (`math.h`)

Przykład:

```
$ gcc -Wall -Wextra -ansi -pedantic -fanalyzer liczba-err.c
```

Debugger w Visual Studio



- punkty przerwania (*breakpoints*)
- krokowanie (*step in*, *step over*, *step out*)
- podgląd wartości zmiennych
- podgląd stosu wywołań

GDB - debuger konsolowy projektu GNU

```
$ gcc -g -o program program.c
$ gdb program
(gdb) run
```

Źródło: ☞ Visual Studio Debugging Seneca, Software testing

- **Analiza statyczna kodu** - analiza struktury kodu źródłowego lub kodu skompilowanego bez jego uruchomienia
- **lint, linter** - program do statycznej analizy
- wykorzystywane w automatyzacji weryfikacji jakości kodu
- zawierają reguły i heurystyki wykrywające podatności
- analiza poprawności składni, detekcja luk bezpieczeństwa, weryfikacja stylu kodu, sugestie dotyczące wydajnośc, ...
- czesto koncentrują się na wybranym zestawie podatności, dlatego warto stosować kilka rozwiązań
- uwaga: mogą generować fałszywie pozytywne ostrzeżenia, czyli błednie wskazywać na istnienie problemu

Przykładowe narzędzia analizy statycznej dla języka C/C++:

- **cppcheck** - głównie wykrywanie krytycznych błędów
  zob.: ☞ lista testów
  ```
  $ cppcheck --enable=all liczba-err.c
  ```

- **clang-tidy** - wiele testów i bogate możliwości konfiguracji
  zob.: ☞ lista testów
  ```
  $ clang-tidy -checks='*'  liczba-err.c
  ```

Narzędzia pozwalające dbać o styl w języku C

- ☞ Artistic Style

```
$ astyle  --style=kr nwd-balagan.c
$ astyle  --style=allman nwd-balagan.c
```

- ☞ ClangFormat

```
$ clang-format -style GNU nwd-balagan.c
```

- on-line ☞ formatter.org ☞ Code Beautify

Przykłady wytycznych dotyczących stylu:

- wytyczne GNU ☞ Making The Best Use of C
- ☞ C Code Style Guidelines
- ☞ Google C++ Style Guide

Git - rozproszony system kontroli wersji służący do zarządzania historią zmian w plikach i kodzie źródłowym projektów

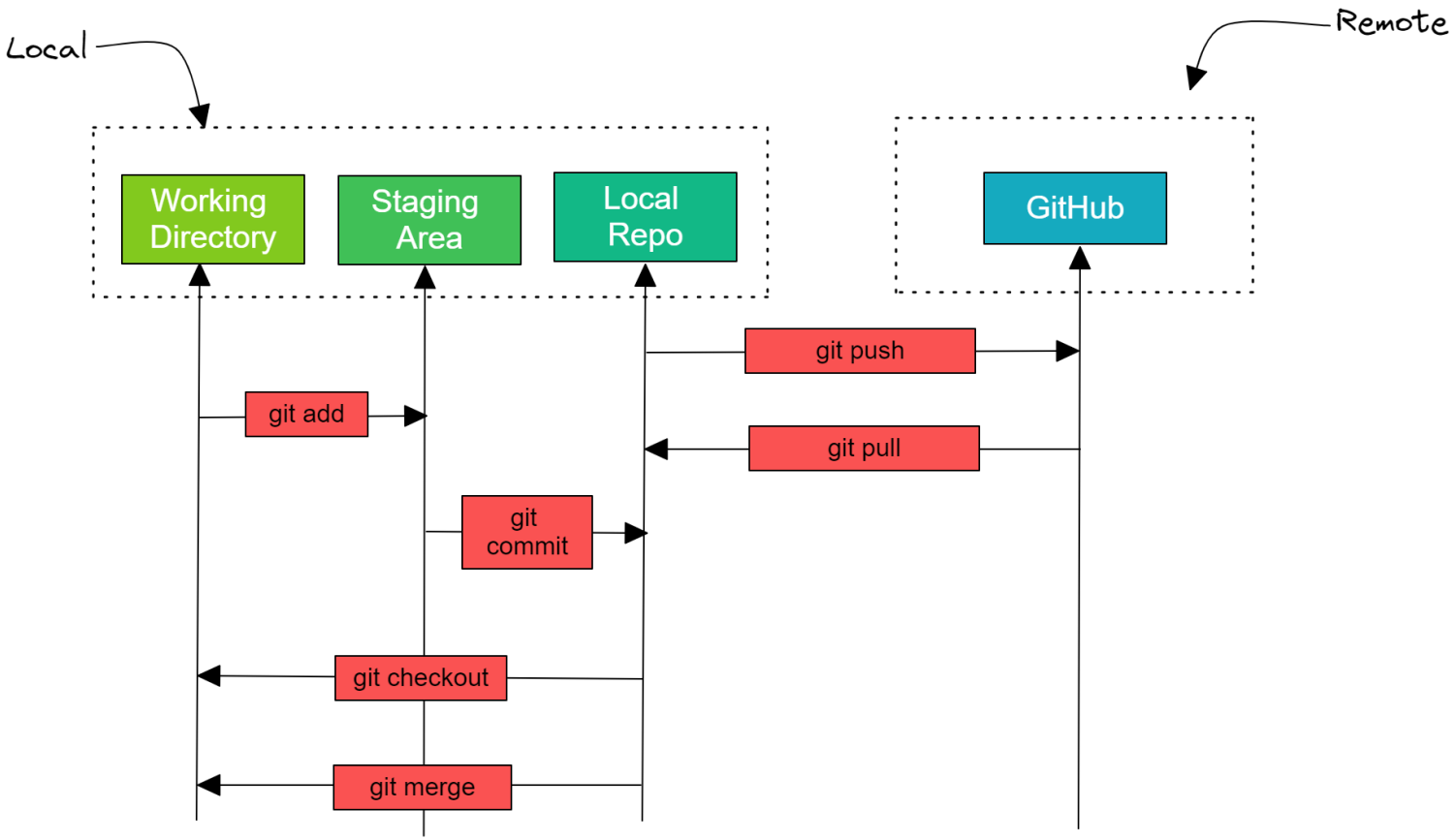
- przechowuje historię zmian w kodzie
- umożliwia współdzielenie kodu w zespołach
- gałęzie, łatwe tworzenie i scalanie gałęzi
- synchronizacja historii pomiędzy repozytoriami
- szybki w działaniu - wiele operacji wykonywanych lokalnie
- elastyczny, wiele aplikacji klienckich, integracja z narzędziami

☞ GitHub - usługa chmurowa udostępniająca repozytoria git
Inne: ☞ GitLab, ☞ BitBucket, ☞ Azure DevOps, ...

Podstawowe komendy git

`add`	dodanie pliku do rewizji (śledzenie zmian)
`commit`	zatwierdzenie zmian, powstaje nowy węzeł w historii repozytorium (lokalnie)
`push`	wypchnięcie zmian do zdalnego repozytorium
`pull`	pobranie zmian ze zdalnego repozytorium i scalenie z lokalną kopią
`merge`	scalanie zmian, łączenie gałęzi
`log`	historia zmian
`diff`	porównywanie zmian w kodzie
`status`	aktualny stan repozytorium

Etapy git



☞ *GitHub and Git Tutorial for Beginners*

Repozytorium z kodami z wykładu:

☞ https://github.com/IS-UMK/pp_wyklad

Klonowanie repozytorium:

```
$ git clone https://github.com/IS-UMK/pp_wyklad
```

Historia zmian:

```
$ git log
```

```
commit 01cc8a5dfb066e280ace0059ad05af9bbc951fbf (HEAD -> master)
Author: Marek <grochu@is.umk.pl>
Date:   Tue Oct 1 11:08:55 2024 +0200

      Dodanie euclid1.f euclid1.pas

commit 0552f871c029186cca1784270411ce77aad5e266
Author: Marek <grochu@is.umk.pl>
Date:   Tue Oct 1 10:51:51 2024 +0200

      Aktualizacja README
```

Dodanie pliku:

```
$ git add nowy.c
```

Status repozytorium:

```
$ git status
```

```
Na gałęzi master
Twoja gałąź jest na bieżąco z „origin/master".

Zmiany do złożenia:
  (użyj „git restore --staged <plik>...", aby wycofać)
nowy plik:       nowy.c
```

Zapamiętanie zmian:

```
$ git commit -m "Opis wprowadzonych zmian"
```

Wypchnięcie zmian do zdalnego repozytorium:

```
$ git push
```

Pobranie i scalenie zmian pochodzących ze zdalnego repozytorium:

```
$ git pull
```