Mittuniversitetet

MID SWEDEN UNIVERSITY

Metoder och verktyg i mjukvaruprojekt

Laboration 2 – Introduktion till CMake

Laborationen syftar till att…
- *… använda byggsystemet Cmake*
- *… använda Git Submodule*
- *… använda git merge*

Innehållsförteckning

Laboration 2

Utvecklingen av en C++ klass för att rita och göra utskrifter till konsolen har delvis spårat ur. Det finns flera versioner med delvis olika implementationer. Din uppgift är att slå samman dessa till en version med alla egenskaper. Utvecklingen började med en implementation där användare kunde sätta en bakgrundsfärg och rensa skärmen. Sen skapades det en ny **branch** med namnet **text**. Där utvecklingen fortsatte och medlemsfunktioner skapades för att skriva ut text inuti en rektangel samt efter en koordinat. Sedermera kom utvecklaren på att det behövdes en medlemsfunktion som ritar ut en rektangel med tecken i en specifik färg på skärmen så denne gick tillbaka till **master-branch** och implementerade detta där. Eftersom både **master** och **text** har utvecklats vidare med flera **commits** är det nu dags för en **merge**.

Uppgift 1

Klona det publika *repositoriet* **screen** till din lokala dator med följande kommando;

```
$ git clone git@bitbucket.org:miun_dt042g/screen.git
```

För att lista alla tillgängliga **branches** används kommandot;

```
$ git branch -a
```

… som kommer att visa;

```
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/text
```

Innan man kan utföra en **merge** måste man ha ett verktyg installerat och det finns många att välja mellan. De som stöds av Git hittar man med följande kommando;

```
$ git mergetool --tool-help
```

Tag hjälp av kursboken och internet för att finna ett **mergetool** som fungerar för ditt operativsystem. För att exempelvis sätta **kdiff3** som default **mergetool** i Git använd följande kommandon;

```
$ git config --global merge.tool kdiff3
```

Du byter alltså ut **`kdiff3`** med det verktyg som du har installerat. Om verktyget inte kan hittas via ovan kommando, måste du antingen lägga till exekveringsfilens sökväg i miljövariabeln **`PATH`** eller manuellt ange detta via följande kommandon;

```
$ git config --global --add merge.tool kdiff3
$ git config --global --add mergetool.kdiff3.path “C:/Dev/Tools/KDiff3/kdiff3.exe”
$ git config --global --add mergetool.kdiff3.trustExitCode false
```

Värdet för **`trustExitCode`** anger huruvida Git skall förlita sig på merge-verktygets redogörelse för en lyckad sammanföring. Här anger vi **`false`** vilket innebär att användaren själv måste bekräfta detta.

Innan vi går vidare bör vi tillföra en .gitignore och i den ange innehåll som skall uteslutas från versionhanteringen. Efter en sammanföring (merge) sparas normalt en backup av originalfiler automatiskt av verktygen och ger dessa filändelsen **`.orig`**. Dessa backupfiler kan vara smidiga vid svårlösta konflikter eller då andra lösningar skall testas, men växer även in antal i takt med projektet tillväxt. Normalt är dessa filer inget man behöver versionshantera eftersom tidigare versioner ändå finns tillgängliga i historiken. Du kan därför med gott samvete ta bort dessa från projektet, men vi skall även berätta för Git att sådana filer alltid skall bortses. Skapa en **`.gitignore`** under projektroten med minst följande innehåll, där vi även utesluter andra onödiga filer som byggsystemet genererar (notera att du ersätter **`.idea/`** med relevant information om du använder annan IDE än CLion):

```
.idea/
cmake-build-debug/
cmake-build-release/
*.orig
```

> *Man kan även i **Git Config** ange att backupfiler aldrig skall genereras via kommandot;*
> ***`git config --global mergetool.keepBackup false`***

Nu är det dags för själva **merge**-momentet. Stå i projektkatalogen och ha **master** som aktiv **branch** och exekvera kommandot;

```
$ git merge origin/text
```

Nu kommer Git att försöka sammanfoga koden som finns i **text** med den som finns i **master**. Lös konflikterna som uppkommer genom att starta **mergetool** med kommandot;

```
$ git mergetool
```

För information hur man utför en **merge** läs manualen för det verktyg som du valt att installera. Spara ändringarna i **mergetool** och sedan i ditt **repository** (med **add** och en **commit**).

Uppgift 2

Nu är det dags att fortsätta utvecklingen. Öppna **CMake**-projektet i din *IDE* och testa att bygga koden. Det visar sig att det saknas ett bibliotek, nämligen **Terminal**. Du skall länka in **Terminal** som en **Git submodule**. Detta görs med kommandot;

```
$ git submodule add git@bitbucket.org:miun_dt042g/terminal.git terminal
```

När detta är klart måste även byggskriptet uppdateras så att **Terminal** läggs till vid kompilering. Detta görs med kommandot;

```
# Add terminal sub directory
add_subdirectory(terminal)
```

Till sist måste man länka med det statiska biblioteket och det görs med kommandot;

```
target_link_libraries(${PROJECT_NAME}
Terminal)
```

Observera! Att `target_link_libraries` måste anropas efter `add_executable`.

Uppgift 3

Nu skall det går att kompilera och köra koden. Testa och kompilera och köra koden i debugläge (med **memstat**) som beskrivs i Lektion 3.

Aj Aj Aj det finns en minnesläcka. Åtgärda den och kompilera om koden!

Skriv om `main`-funktionen och skapa ett fönster av standardstorlek `80x24`. Nu är det dags att vara lite kreativ genom att rita och skriva ut text i olika färger. Enda kravet är att ni måste anropa alla publika medlemsfunktioner i `Screen` minst en gång.

När detta är klart sparar du koden och gör en **commit** med lämpligt meddelande. Lägg sedan till en `README.md` under projektroten där du skriftligt redovisar tillvägagångssätten under laborationsarbetet och gör en ny **commit** för denna fil. Uppdatera sedan fjärradressen **origin** att peka till ditt aktuella studentrepo i kursens Bitbucket Team;

```
$ git remote remove origin
$ git remote add origin git@bitbucket.org:miun_dt042g/studid_laboration_2_vt21.git
```

Synkronisera så att alla ändringar blir synliga på *Bitbucket*!

Redovisning

Utför en formell inlämning i avsedd inlämningslåda i kursens Moodle-instans. Projekttillstånd och nödvändiga filer hämtas direkt från ditt dedikerade studentrepo, så se till att detta repo är uppdaterat innan inlämningen.

Arbetet skall bedrivas enskilt med individuell redovisning, och pararbete är inte tillåtet!