Snabbgenomgång av versionshantering av kod

Kapitel 1: En Introduktion till Git

Vad är Git?

Git är ett distribuerat versionshanteringssystem som används för att spåra ändringar i källkoden under utvecklingsprocessen. Det skapades av Linus Torvalds år 2005 och har sedan dess blivit standard inom många utvecklingsprojekt.

Varför använda Git?

Git erbjuder flera fördelar, inklusive:

- **Versionskontroll:** Git sparar och hanterar alla ändringar i koden, vilket möjliggör enkel återgång till tidigare versioner om något går fel.
- **Distribuerad utveckling:** Git tillåter flera utvecklare att arbeta parallellt på olika delar av projektet, vilket ökar effektiviteten.
- **Branching och merging:** Utvecklare kan skapa separata grenar (branches) för att experimentera med nya funktioner eller fixa buggar utan att påverka huvudkoden. Dessa grenar kan sedan enkelt slås samman (merged) när de är klara.

Grundläggande Git-kommandon

Här är några grundläggande Git-kommandon för att komma igång:

Initialiserar ett nytt Git-repositorium i den aktuella katalogen.

```
git init
```

Klonar ett befintligt Git-repositorium från en given URL.

```
git clone [URL]
```

Lägger till en fil i den så kallade "staging area" för att förbereda för en commit.

```
git add [fil]
```

Sparar ändringarna i koden med ett beskrivande meddelande.

```
git commit –m "Meddelande"
```

Hämtar och integrerar ändringar från ett fjärrrepositorium till det lokala.

```
git pull
```

Hämtar ändringar från ett fjärrrepositorium till det lokala repositoriet. Det viktiga att notera är att **git fetch** inte automatiskt sammanfogar (merge) de hämtade ändringarna med den aktuella grenen. Istället uppdateras endast den lokala referensen till de fjärranlagda grenarna.

```
git fetch
```

Skickar ändringar från det lokala repositoriet till det fjärranlagda repositoriet.

```
git push
```

Visar befintliga grenar och den aktuella grenen.

```
git branch
```

Slår samman ändringar från en specifik gren till den aktuella grenen.

```
git merge [gren]
```

Detta är bara några grundläggande kommandon. Nästa kapitel kommer att utforska GitLab, en plattform som bygger på Git för att underlätta samarbete och projektadministration.

Kapitel 2: GitLab - En Plattform för Kollaborativ Utveckling

Vad är GitLab?

GitLab är en webbplattform som bygger på Git och erbjuder en omfattande uppsättning verktyg för kollaborativ utveckling. Den ger utvecklare möjlighet att hantera källkodsrepository, spåra ärenden (issues), genomföra kodgranskningar och mer, allt inom en enda plattform.

Funktioner och Fördelar med GitLab:

- **Repository Hantering:** Skapa, hantera och samarbeta kring Git-repositorier på ett enkelt sätt. GitLab erbjuder funktioner som förgreningshantering, pull-requests och automatiska byggen.
- **Kontinuerlig Integration (CI) och Kontinuerlig Leverans (CD):** Automatisera bygg- och testprocesser för varje ändring i koden. Detta hjälper till att upptäcka och åtgärda problem tidigt.
- **Ärendehantering:** Spåra och hantera projektuppgifter, buggar och förbättringsförslag med hjälp av GitLabs inbyggda ärendehanteringssystem.

- **Kodgranskning:** Utför kodgranskningar genom att skapa och granska pull-requests. Detta främjar samarbete och säkerställer högkvalitativ kod.
- **Wiki och Dokumentation:** Skapa och underhåll dokumentation direkt inom GitLab för att underlätta kunskapsdelning inom teamet.
- **Access Control:** Hantera åtkomstnivåer och behörigheter för att säkerställa att endast auktoriserade användare kan utföra specifika åtgärder.

Komma lgång med GitLab:

- **Skapa ett konto:** Besök GitLab-webbplatsen och skapa ett konto om du inte redan har ett.
- **Skapa ett projekt:** Efter inloggning kan du skapa ett nytt projekt och koppla det till ett Git-repositorium.
- **Lägg till Medarbetare:** Bjud in teammedlemmar till ditt projekt och ställ in behörigheter för varje användare.
- **Skapa en Gren (Branch):** Använd grenar för att isolera och arbeta på nya funktioner eller fixa buggar utan att påverka huvudkoden.
- **Skapa en Pull-Request:** När du är klar med dina ändringar, skapa en pull-request för att föreslå ändringarna till huvudgrenen.
- **Kontinuerlig Integration (CI):** Konfigurera CI-pipelines för att automatisera byggen och tester vid varje ändring i koden.

GitLab erbjuder en mängd funktioner för att underlätta och förbättra samarbetet inom utvecklingsteam. Nästa kapitel kommer att fokusera på GitHub Desktop, en användarvänlig Git-klient för desktop-miljöer.

Kapitel 3: GitHub Desktop - En Enkel Git-Klient för Desktop

Vad är GitHub Desktop?

GitHub Desktop är en användarvänlig Git-klient som syftar till att göra det enklare för utvecklare att använda Git utan att behöva använda kommandoraden. Den ger en grafisk användargränssnitt (GUI) för att hantera Git-repositorier och genomföra grundläggande Git-åtgärder.

Funktioner och Fördelar med GitHub Desktop:

- **Enkel att Använda:** GitHub Desktop erbjuder en intuitiv och användarvänlig gränssnitt som gör det lätt att komma igång med Git.
- **Visuell Historik:** Se och hantera historiken över dina ändringar, vilket gör det enkelt att återgå till tidigare versioner av koden.
- **Grenhantering:** Skapa, byt och sammanfoga grenar enkelt med hjälp av den visuella gränssnittet.
- **Enkel Commit och Push:** Förbered och genomför Git-commits och pushes med några enkla knapptryckningar.
- Konfliktlösning: Hantera konflikter i koden direkt via GitHub Desktop för att undvika problem under sammanfogning.
- **GitHub Integration:** Enkelt koppla GitHub Desktop till ditt GitHub-konto för att synkronisera dina projekt.

Komma lgång med GitHub Desktop:

- **Installation:** Ladda ner och installera GitHub Desktop från GitHub Desktop-webbplatsen.

- **Logga in:** Använd ditt GitHub-konto för att logga in i GitHub Desktop.
- **Klona Ett Projekt:** Klona ett befintligt projekt från ditt GitHub-konto eller lägg till ett lokalt projekt.
- **Arbeta med Grenar:** Skapa och byt mellan grenar för att isolera och utveckla nya funktioner.
- **Commit och Push:** Förbered ändringar, gör commits och pusha dem till ditt Git-repositorium med några enkla steg.
- **Visuell Historik:** Utforska och hantera historiken över ditt projekt för att förstå ändringar över tiden.

GitHub Desktop är ett utmärkt alternativ för de som föredrar att arbeta med Git genom ett grafiskt gränssnitt. Genom att kombinera verktyg som GitLab och GitHub Desktop kan du effektivt hantera ditt projekt och underlätta samarbete inom ditt utvecklingsteam.

Kapitel 4: Länka GitLab Repository till Azure DevOps

Att länka ditt GitLab-repositorium till Azure DevOps möjliggör integrerad hantering av projekt, hantering av arbetsuppgifter, och implementering av CI/CD (Continuous Integration/Continuous Deployment) genom Azure DevOps tjänster. Här är en steg-för-steg guide för att genomföra detta:

Steg 1: Skapa ett Projekt i Azure DevOps

1. Gå till Azure DevOps-webbplatsen och logga in med ditt konto.
2. Skapa ett nytt projekt genom att klicka på "New Project" och följ instruktionerna för att ange projektnamn och inställningar.

Steg 2: Skapa ett GitLab Personal Access Token

1. Logga in på GitLab och gå till "Settings" -> "Access Tokens".
2. Skapa ett nytt personligt access token med behörighet att läsa repositories.
3. Kopiera tokenet eftersom det kommer att användas senare för autentisering.

Steg 3: Skapa en Service Connection i Azure DevOps

1. Inne i ditt Azure DevOps-projekt, gå till "Project Settings" -> "Service connections" -> "New service connection" -> "GitLab".
2. Ange GitLab-webbadressen och det tidigare skapade personal access token.
3. Klicka på "Verify and save" för att bekräfta anslutningen.

Steg 4: Skapa en YAML-fil för CI/CD

I ditt GitLab-repositorium, skapa en fil med namnet .gitlab-ci.yml i roten. Konfigurera YAML-filen med steg för att bygga, testa och distribuera din kod.

Exempelvis:

```
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
```

```
  script:
    - echo "Building the project"

test_job:
  stage: test
  script:
    - echo "Running tests"

deploy_job:
  stage: deploy
  script:
    - echo "Deploying to production"
```

Steg 5: Aktivera GitLab CI/CD-pipelinen

1. När du gör en push till ditt GitLab-repositorium, kommer GitLab att automatiskt upptäcka .gitlab-ci.yml och starta CI/CD-pipelinen.
2. Du kan övervaka pipelinen och dess steg direkt från GitLab-gränssnittet.

Genom att följa dessa steg skapar du en sömlös integration mellan ditt GitLab-repositorium och Azure DevOps för att hantera projektfaser och automatisera din CI/CD-process. Detta underlättar en effektiv och strukturerad utvecklingscykel.