

# Gitflow

I denna labb tar vi ännu ett steg framåt i vår förmåga att arbeta med Git och Github. Vi kommer därför att arbeta enligt "Gitflow", vilket följer en väldefinierad struktur med klara och tydliga regler kring hur vi arbetar tillsammans som ett team.

## Gitflow-flöde

Följande är en överblick över Gitflow-flödet, mer detaljer kommer nedan:

- I. Skapa develop-branch från main
- II. Skapa feature-branches från develop
- III. När en feature är "färdig", mergea featuren till develop
- IV. När en release är klar, mergea develop till main
- V. Om det uppstår problem på main, skapa hotfix-branch från main
- VI. När hotfix är klar, mergea:
  - a. Hotfix till main
  - b. Hotfix till develop

# Branches

## Features

Dessa branches skapas från develop-branchen. Viktigt att notera här är att vi ska sluta tänka på en branch som en "person". En feature kan arbetas på av mer än en person, men det är viktigt att ni vet vem som jobbar med vad och att ni inte arbetar i samma filer och på samma sak. **Kommunikation** är av yttersta vikt.

## Develop

Develop-branchen agerar som en integrationsbranch för features. Kod som merge:as hit ska innehålla färdiga features som testats genom lokal körning samt er Feature-pipeline. Även denna branch ska skyddas genom att minst 1 annan medlem i gruppen ska godkänna koden efter en code review.

När develop-branchen har tillräckligt med nya och välfungerande features är det dags att mergea till main. Lägg inte till några nya features till develop när en ny release-cykel påbörjats (alltså när ni är redo att mergea med main), ni ska endast utföra eventuella bugfixes och se till att allt är uppdaterat och redo. Detta innebär att ni har gjort klart alla tester för exempelvis "Kim och Familjen"-testfallet och då lägger upp detta på main.

## Main

Ska skyddas genom att alla i gruppen behöver godkända Pull Requests. Ska endast innehålla färdig kod som gått igenom alla era rigorösa tester.

## Hotfixes

Förhoppningsvis behövs inte detta alls, men om det uppstår problem efter att ni har merge:at till main och alla Pull Requests gått igenom behöver det såklart fixas. Skapa en hotfix-branch ifrån er main-branch som ni arbetar på. Fixa problemen innan ni fortsätter med annat. När ni löst problemen, merge:a med main och develop igen så att ni ser till att allt är uppdaterat!

# Pipelines

## Feature

Denna pipeline triggas av att ni lägger till en ny feature. Detta åstadkoms genom att ni använder Robot Framework-tags på era nya features och att ni följer namnkonventioner på era feature-branches.

## Develop

När en feature har gått igenom lokala körningar samt tester i Features-pipelinan är den (om allt gått bra) redo att läggas till i develop-branchen. Denna pipeline kan ni se som att den utför integrationstestning. Er feature-pipeline testar enskilda features (unit testing) medan develop-branchen testar exempelvis ett helt test case (som nu fått nya features)

## Main

Triggas av att ny kod mergeas med main-branchen. Denna pipeline köra alla tester så att ni vet att allt fungerar som ni tänkt.

# Instruktioner - Uppgift

1. Gå in på <https://dev.azure.com/mt24-labb2/> och det projekt som er grupp blivit tilldelad.
2. Lägg till yaml-filerna från iths-distans i ert repo
3. Skapa branches för main och develop
4. Skapa 3 olika pipelines i Azure DevOps: "main", "develop" och "features", varje pipeline kopplas till den motsvarande yaml-filen.
5. Notera att det finns 4 yaml-filer. "install-dependencies" ska läggas i en mapp namngiven "templates"
6. För att kunna köra tester på vår lokala hemsida (Jurasstina-Kalle Park) krävs det att vi kör vår testsvits browser som "headless" (exempelvis för chrome skriver ni då "headlesschrome" istället för "chrome")
7. Skapa en ny folder i results/ i ert repo, döp den nya foldern till "logs" (alltså är pathen results/logs)
8. Med hjälp av Gitflow-arbetsättet ska ni nu refaktorera koden så att den uppfyller kraven som ställs i avsnittet "Filstruktur". Glöm inte att hålla er kod uppdaterad för att undvika merge conflicts

## Filstruktur

- 1 fil för varje testsvit, exempelvis kim.robot och stinapalle.robot, dessa ska finnas i en folder som heter "tests"
  - Dessa innehåller endast testfall som använder "Given, When, Then"-syntax och nödvändiga Settings (Library, Resource, Test Setup och Teardown)
  - Se till att ni inte använder absoluta paths, använd \${EXECDIR} som utgångspunkt
- 1 keyword-fil för varje testsvit (innehåller de keywords som era "Given, When, Then"-tester använder), exempelvis kim\_keywords.robot, stinapalle\_keywords.robot, dessa läggs i resources/keyword\_files/
- Keyword-fil som innehåller generella keywords ("keywords.robot" exempelvis), vilka används av era svit-specifika keywords
- Eventuella Python-filer för variabler, utökad funktionalitet etc (placeras i relevant folder, exempelvis "util" eller "page\_object\_model")
- Testrapporter från Robot Framework hamnar i en egen folder (results/logs/)

# AI-instruktioner

1. Kopiera era yaml-filer och be en LLM (AI "Chatbot", exempelvis Chat-Gpt, Claude, Mistral etc):
  - a. Förklara innehållet i yaml-filen
  - b. Optimera koden
2. Skapa en ny chat med en LLM (gärna en annan modell/leverantör):
  - a. Be den läsa igenom en av era testsviter men instruera den specifikt att bara verifiera att den läst (se videogenomgång på iths-distans och föreläsning)
  - b. Förklara 2-3 delar (exempelvis ett test case eller en av koden till chat-boten och be den förklara om ni tänker rätt eller fel i er förklaring)
    - i. Bonus (ej obligatoriskt): Förklara något som ni vet är fel, se om den rättar er eller ej

# Videoinstruktioner

Videolängd: ca 5 minuter

1. Spela in en kort video (skärminspelning) där ni visar er konversation med AI och gör en kortfattad analys av svaren. Precis som i Labb 1 vill jag att alla i gruppen pratar i videon och att ni presenterar er med namn när ni pratar.
  - a. Exempel på analys för videon: Var svaren bra? Fanns det felaktigheter? Vad lärde ni er? Hade ni förstått er kod rätt, förstod AI koden?
  - b. Svara även i videon på om ni använde AI i någon del av er utvecklingsprocess. Varför? Varför inte?
2. I samma video ska varje medlem i gruppen besvara följande frågor:
  - a. Vad var svårast med labben?
  - b. Förklara kortfattat hur ni arbetade enligt "Gitflow" under er refaktoreringsprocess
  - c. Vad hade du velat lära dig mer eller förstå bättre kopplat till Azure DevOps eller CI/CD generellt?

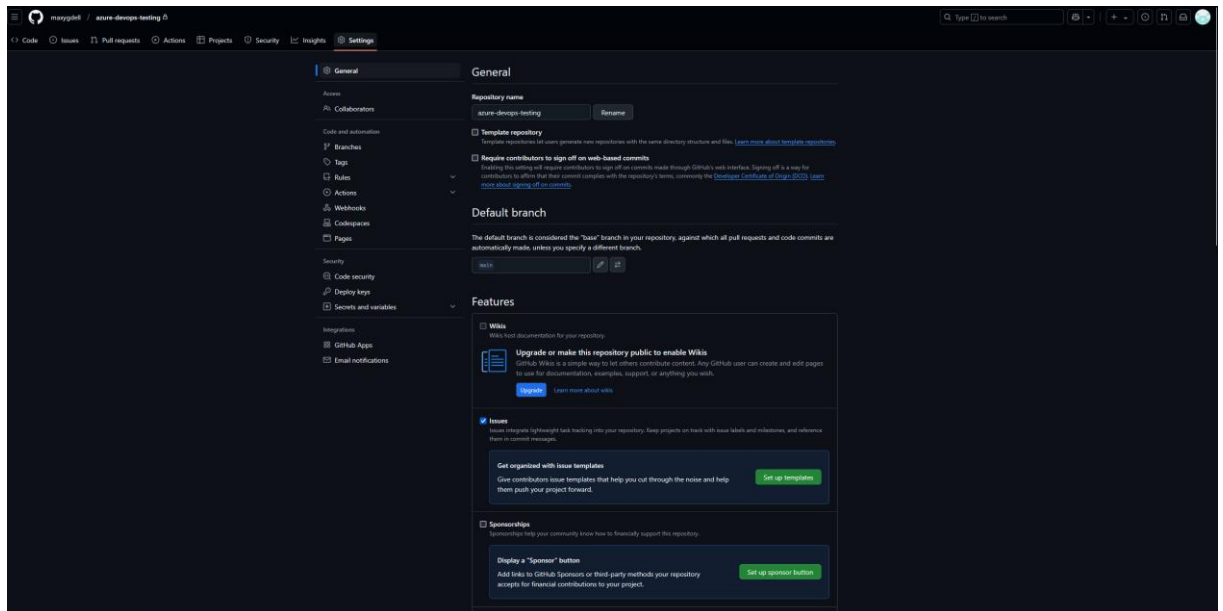
# Allmänna tips

1. Bestäm **från start** vem som jobbar med vad (dela upp det i features), arbeta inte i samma filer.
2. Om ni ska arbeta med en fil som kräver att en mer generell fil (variables.py, keywords.robot etc) förändras. Bestäm då att ni gör detta tillsammans så att förändringar inte tar sönder er kodbas.
3. Ha inte några mellanslag i era filnamn, det kan orsaka problem då ni behöver kunna använda script, vilka exekveras i terminalen.
4. Mer info om hur en pipeline i Azure DevOps byggs upp finns här:  
<https://learn.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops>
3. Kolla vad som finns förinstallerat på ubuntu-latest så att era pipelines inte tar längre tid att köras i onödan. Mer information finns här:  
<https://github.com/actions/runner-images/blob/main/images/ubuntu/Ubuntu2204-Readme.md>
4. Glöm inte ta hänsyn till paths. Paths fungerar annorlunda på en pipeline jämfört med era lokala maskiner. Ett tips är att använda `${EXECUTOR}`, då utgår ni ifrån "root"-folder.
5. För att få tillgång till de skapade rapporterna måste dessa publiceras som en artifact i Azure DevOps. Om ni är intresserade kan ni kan läsa mer om artifacts här: <https://learn.microsoft.com/en-us/azure/devops/pipelines/artifacts/build-artifacts?view=azure-devops&tabs=yaml>
6. Om ditt bygge fallerar, försök att läsa vad för felmeddelande ni får och vid vilket tillfälle. Om ni inte kan förstå er på det själva har Microsoft en ovärderlig resurs (engelska): Common Issues - <https://learn.microsoft.com/en-us/azure/devops/pipelines/troubleshooting/troubleshooting?view=azure-devops#common-issues>

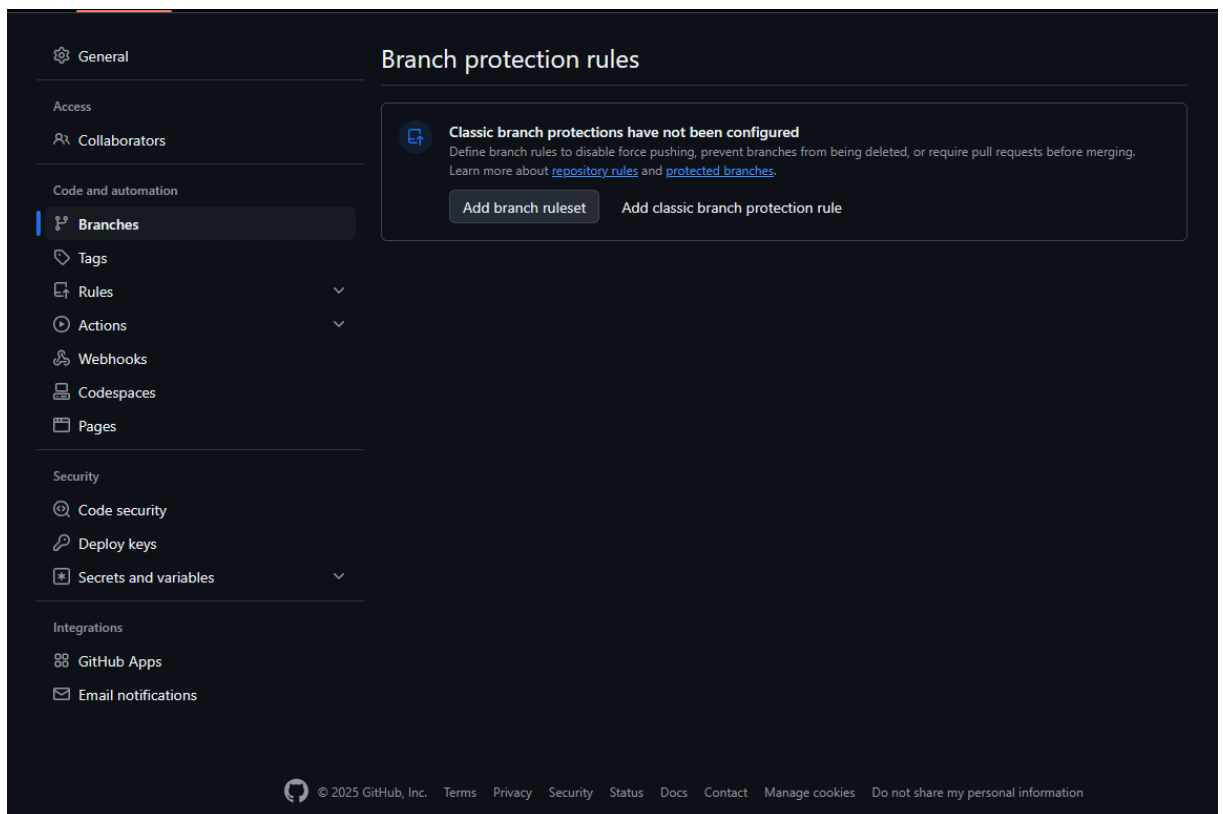
# Branch Protection Rules

För att kunna säkerställa att kod som läggs till i viktiga branches först går igenom en ”code review” behöver ni applicera ”Branch Protection Rules”. Detta kan ni göra direkt i ert repo på Github genom följande steg:

## 1. Gå till Settings




## 2. Klicka på Branches, sedan ”Add branch ruleset”




3. Ge ditt ruleset ett namn, välj "Active" på Enforcement status och lägg till "Default" i Branch targeting criteria. Det ska se ut som på följande bild.

Rulesets / New branch ruleset

 **Protect your most important branches**


[Rulesets](#) define whether collaborators can delete or force push and set requirements for any pushes, such as passing status checks or a linear commit history.

 Your rulesets won't be enforced on this private repository until you [move to GitHub Team organization account](#).

Ruleset Name \*

Main Branch

Enforcement status

 Active

Bypass list + Add bypass

Exempt roles, teams, or apps from this ruleset by adding them to the bypass list.

Bypass list is empty



Targets

Which branches do you want to make a ruleset for?

Target branches

Branch targeting determines which branches will be protected by this ruleset. Use inclusion patterns to expand the list of branches under this ruleset. Use exclusion patterns to exclude branches.

Branch targeting criteria Add target

 Default 



4. Gå nu ner till Rules och Branch Rules. Bocka i ”Require a pull request before merging”. För main-branchen ska ni (i en grupp med 3 personer) ha 2 required approvals.

## Rules

Which rules should be applied?

### Branch rules

☐ **Restrict creations**  
Only allow users with bypass permission to create matching refs.

☐ **Restrict updates**  
Only allow users with bypass permission to update matching refs.

☒ **Restrict deletions**  
Only allow users with bypass permissions to delete matching refs.

☐ **Require linear history**  
Prevent merge commits from being pushed to matching refs.

☐ **Require deployments to succeed**  
Choose which environments must be successfully deployed to before refs can be pushed into a ref that matches this rule.

☐ **Require signed commits**  
Commits pushed to matching refs must have verified signatures.

☒ **Require a pull request before merging**  
Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.  

Hide additional settings ^

#### Required approvals

2 ▾

The number of approving reviews that are required before a pull request can be merged.

☐ **Dismiss stale pull request approvals when new commits are pushed**  
New, reviewable commits pushed will dismiss previous pull request review approvals.

☐ **Require approval of the most recent reviewable push**  
Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

☐ **Require conversation resolution before merging**  
All conversations on code must be resolved before a pull request can be merged.

☐ **Request pull request review from Copilot** Preview  
Automatically request review from Copilot for new pull requests, if the author has access to Copilot code review.

#### Allowed merge methods

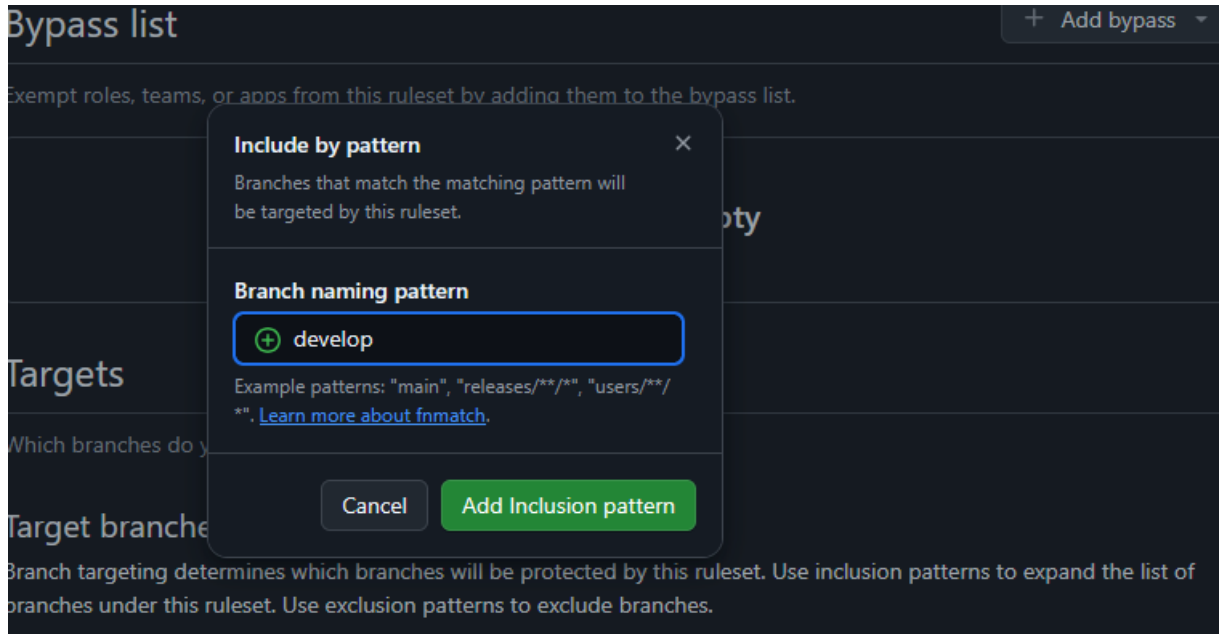
Preview

Merge, Squash, Rebase ▾

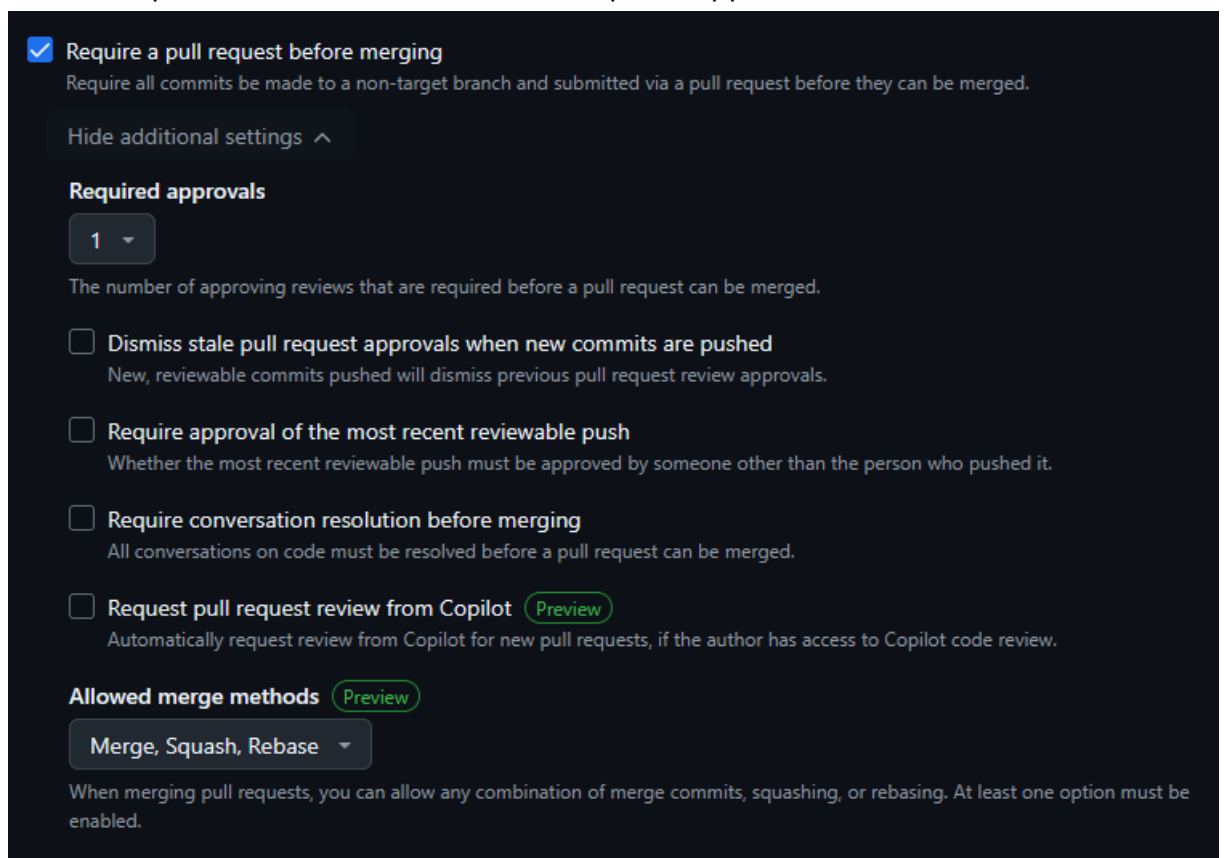
When merging pull requests, you can allow any combination of merge commits, squashing, or rebasing. At least one option must be enabled.

☐ **Require status checks to pass**  
Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.

5. Spara och skriv in ditt lösenord. Notera att om ert projekt är inställt som "private" så kommer ni (förmodligen) inte kunna applicera dessa regler, så ha ert repo publikt!
6. Vi gör liknande sak på develop-branchen, med ett nytt namn. På Add Target väljer ni nu istället include by pattern.



7. För develop-branchen kan ni istället ha 1 required approval



8. Glöm inte spara!

# Sätta upp pipelines

Ni har under tidigare föreläsning (CI-CD-MT24 på ITHS-distans) sett hur ni skapar en pipeline, men vad gör det, om ni inte vet vad som ska skrivas i er yaml-fil? Låt oss kika närmre på det! Jag kommer ge er filer för 3 olika pipelines, men jag vill att ni läser och förstår dessa filer.

## Feature-pipeline

Följande fil heter "feature-pipeline.yml" och innehåller kod vars trigger är feature/\* Detta innebär att skriptet är uppsatt för att köras när en branch som har namnet feature/{*något\_namn\_här*} uppdateras. Det viktiga här är skriptet som finns på rad 13-18.

```
! feature-pipeline.yml
1  trigger:
2    - feature/*
3
4  pool:
5    vmImage: 'ubuntu-latest'
6
7  steps:
8    # Install dependencies (Robot Framework, Selenium Library)
9    - script: |
10      pip install --upgrade robotframework-seleniumlibrary
11      displayName: 'Install Robot Framework and dependencies'
12
13    - script: |
14      robot --include new-feature
15      --output results/logs/output_feature_test.xml
16      --log results/logs/log_feature_test.html
17      --report results/logs/report_feature_test.html tests/kim.robot
18      displayName: 'Run Tests tagged with new-feature'
19
20    # Publish the Robot Framework reports as pipeline artifacts.
21    - task: PublishPipelineArtifact@1
22      inputs:
23        targetPath: '$(System.DefaultWorkingDirectory)/results/logs/'
24        artifactName: 'robot-reports'
25        displayName: 'Publish Robot Framework Reports'
26
```

--include new-feature säger att alla tester som har taggats med "new-feature" ska köras.

Exempelvis:

```
I should be able to buy weekend safari tickets
[Tags]    new-feature
Book weekend Safari
```

Notera att vi i slutet (rad 17) har tests/kim.robot, vilket betyder att detta skript enbart kommer köra filen kim.robot. För att köra en hel folder (exempelvis foldern test) är det bara skriva pathen till foldern, i detta exempel "tests/"

## Develop-pipeline

```
! develop-pipeline.yml
1  trigger:
2    - develop
3
4  pool:
5    vmImage: 'ubuntu-latest'
6
7  steps:
8    # Install dependencies (Robot Framework, Selenium Library)
9    - script: |
10      pip install --upgrade robotframework-seleniumlibrary
11      displayName: 'Install Robot Framework and dependencies'
12
13    # Scripts for running all tests in a file and to run tests based on test case name (integration testing)
14    - script: |
15      robot --output results/logs/output_kim_test.xml --log results/logs/log_kim_test.html
16      --report results/logs/report_kim_test.html tests/kim.robot
17      displayName: 'Run Test File "Kim"'
18
19    - script: |
20      robot --test "Buying Safari Tickets For The Family"
21      --output results/logs/output_kim_test_safari.xml
22      --log results/logs/log_kim_test_safari.html
23      --report results/logs/report_kim_test_safari.html tests/
24      displayName: 'Run Specific Tests'
25
26    # Publish the Robot Framework reports as pipeline artifacts.
27    - task: PublishPipelineArtifact@1
28      inputs:
29        targetPath: '${System.DefaultWorkingDirectory}/results/logs/'
30        artifactName: 'robot-reports-develop'
31      displayName: 'Publish Robot Framework Reports'
```

Integrationstestning. Så vi kör Test Cases (Buying Safari Tickets For The Family) och en specifik fil (kim.robot).

## Main-pipeline

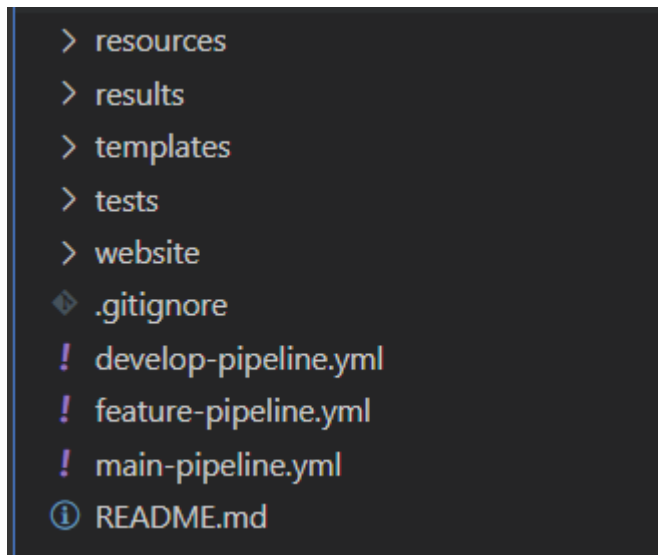
```
! main-pipeline.yml
1  trigger:
2    - main
3
4  pool:
5    vmImage: 'ubuntu-latest'
6
7  steps:
8    # Install dependencies (Robot Framework, Selenium Library)
9    - script: |
10      pip install --upgrade robotframework-seleniumlibrary
11      displayName: 'Install Robot Framework and dependencies'
12
13    # Run all tests
14    - script: |
15      robot --output results/logs/output_all_test.xml
16      --log results/logs/log_all_test.html
17      --report results/logs/report_all_test.html tests/
18      displayName: 'Run All Tests'
19
20    # Publish the Robot Framework reports as pipeline artifacts.
21    - task: PublishPipelineArtifact@1
22      inputs:
23        targetPath: '$(System.DefaultWorkingDirectory)/results/logs/'
24        artifactName: 'robot-reports-main'
25        displayName: 'Publish Robot Framework Reports'
26
```

När vi mergear med main triggas vår main-pipeline.yml-fil och kör alla tester.

# Repostruktur - bilder

Ert repo bör ha en struktur som är tydligt uppdelat, se följande bilder som exempel.

Översikt



Expanderade folders:

