POWERING
POTENTIAL

DevOps Essentials
Part 1 – Git & Version Control

2022 Version

PRINCE2®
TRAINING ORGANIZATION ACCREDITED BY
PEOPLECERT ON BEHALF OF AXELOS

Housekeeping

Course Outline

- **Git Deep-Dive**
  - History of Version Control
  - Who, when, what and why
  - Git Commands Overview
  - Using a branch
  - Working as a Team

COURSE OUTCOMES

- Demonstrate an understanding of the source control usage.
- Apply Git commands to demonstrate activities required to build a solution.
- Work together as a DevOps Team.

WHAT IS VERSION CONTROL?

- Version Control is the software system capable of tracking all changes to the code.
- As the product evolves, multiple stable or in-progress versions are created.
- It helps product development teams by maintaining multiple versions of the code.

WHAT IS VERSION CONTROL?

- You can also revert to a previous version if any developer makes a mistake or if a deployment goes wrong.
- Popular examples: GitHub, BitBucket, Microsoft Team Foundation Server.

HISTORY OF VERSION CONTROL

- 1972: First version control Source Code Control System (SCCS) developed by Marc Rochkind at Bell Laboratories
- 1982: Revision Control System
- 1986: Concurrent Versions System with added support for change tracking
- 2000: Subversion introduces directory-level changes as opposed to file-level
- 2010: Microsoft Team Foundation Server brings in automated testing

WHY USE VERSION CONTROL?

- Global collaboration with increased visibility.
- Accelerated product delivery.
- Traceability for all changes ever made.

WHY USE VERSION CONTROL?

- Concurrent development for complex projects with multiples versions of code and files.
- Automating tasks such as testing and development leading to higher quality and increased productivity.
- Uninterrupted availability and disaster recovery.

A STORY

The Random Bank has just finished building a loan-risk calculator for their loan applicants. Their engineering head, Jane, is looking to add new functionality like extracting credit scores and requesting the supporting documents from the applicants.

The Random Bank also wishes to extend this desktop application to Web and Mobile apps.

Jane already has a team of developers, testers and designers who will collaborate to enhance the application. They have decided to use GitHub, a free Version Control System to ensure a smooth build and release lifecycle.

Activity: (15 mins)
Create GitHub Account

- Work individually, or in groups of 2 or 3.
- Navigate to www.github.com on your web browser.
- Create an account.
- Create a new public repository.

BUT BEFORE YOU GET STARTED

Let’s have a quick look at some MUST-KNOW commands!

These commands will then help Jane and her team to update their application.

GIT COMMANDS

Git is a distributed version-control system that keeps tracks of changes for any set of files. There are various commands in Git widely used in the software industry. Here are some frequently used commands that you ought to know!

GETTING STARTED

An Overview

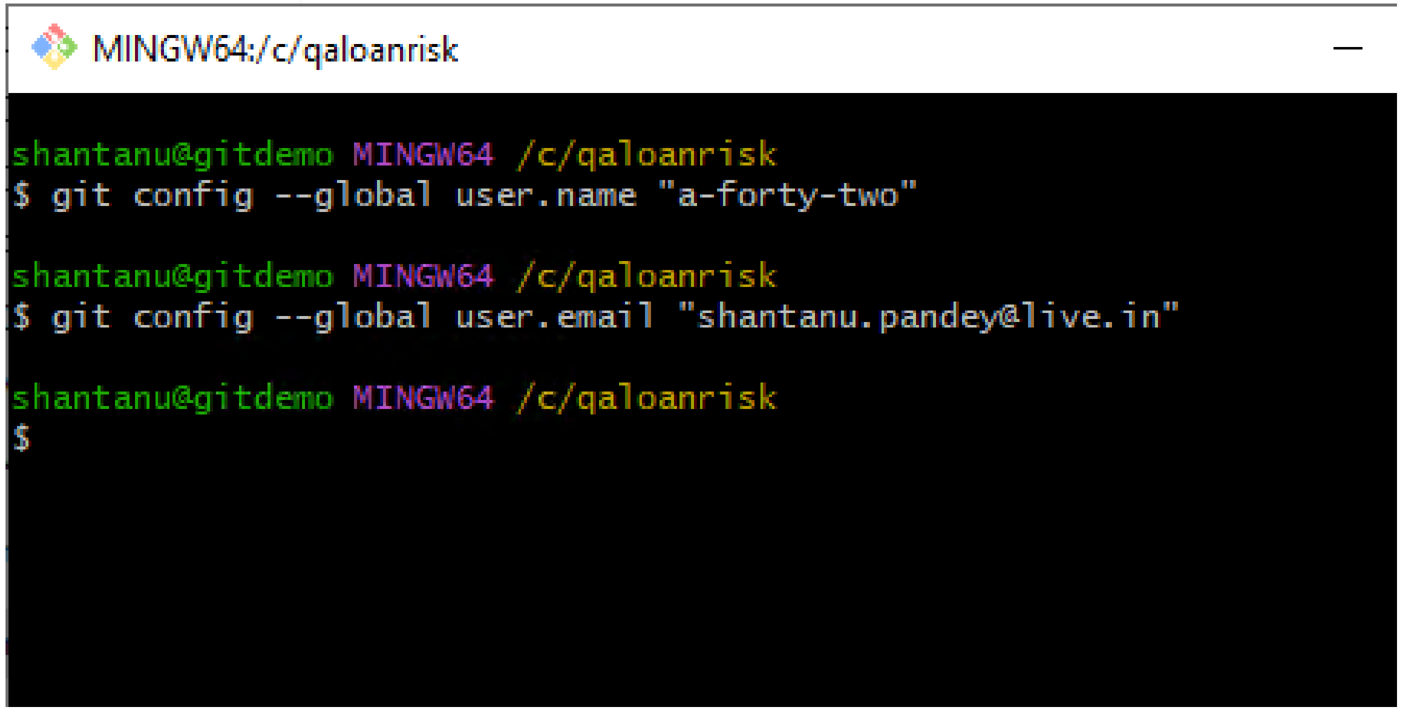
1. Download and install Git.
2. Create a local Git repository.
3. Create a remote Git repository.
4. Add the file(s) to the local repository.
5. Review the changes (or unstage).
6. Create a commit.
7. Work as a team! Branch out.
8. Merge the changes.
9. Conflict resolution.

SETUP THE USER

Configure the Github account with the local system, using *git config* commands:

```
git config --global user.name "your_username"

git config --global user.email
"your_email_address@example.com"
```

SETUP THE USER



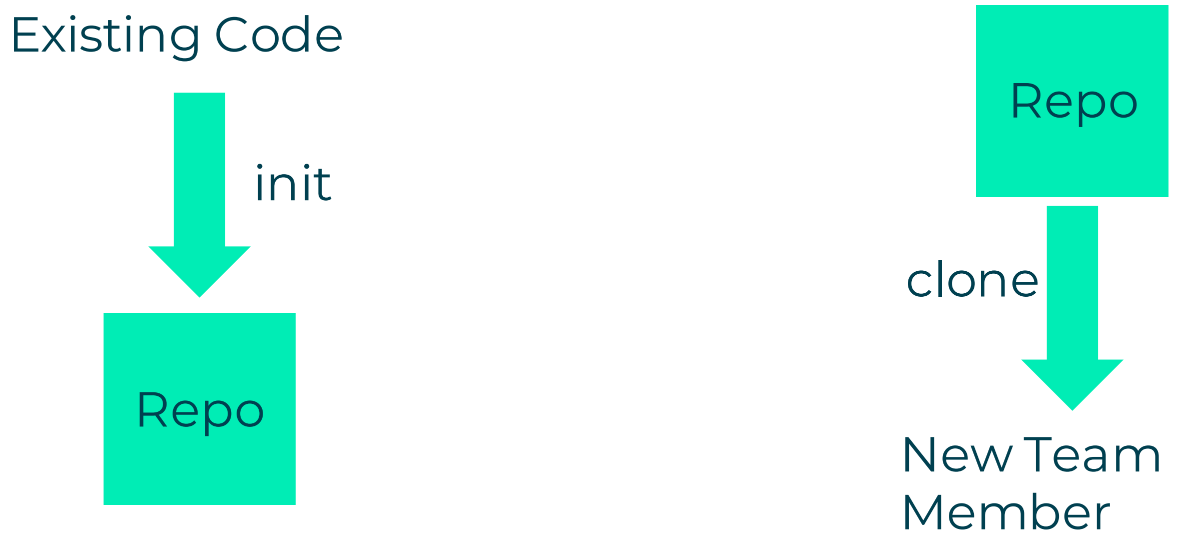
```
shantanu@gitdemo MINGW64 /c/qaloanrisk
$ git config --global user.name "a-forty-two"

shantanu@gitdemo MINGW64 /c/qaloanrisk
$ git config --global user.email "shantanu.pandey@live.in"

shantanu@gitdemo MINGW64 /c/qaloanrisk
$
```

Activity: (10 mins) Configure Local Account

- Work individually, or in groups of 2 or 3.
- Download and install Git.
- Open Git Bash.
- Configure your git account.

2 WAYS TO GET STARTED

Either create a NEW local repository, or clone an existing repository

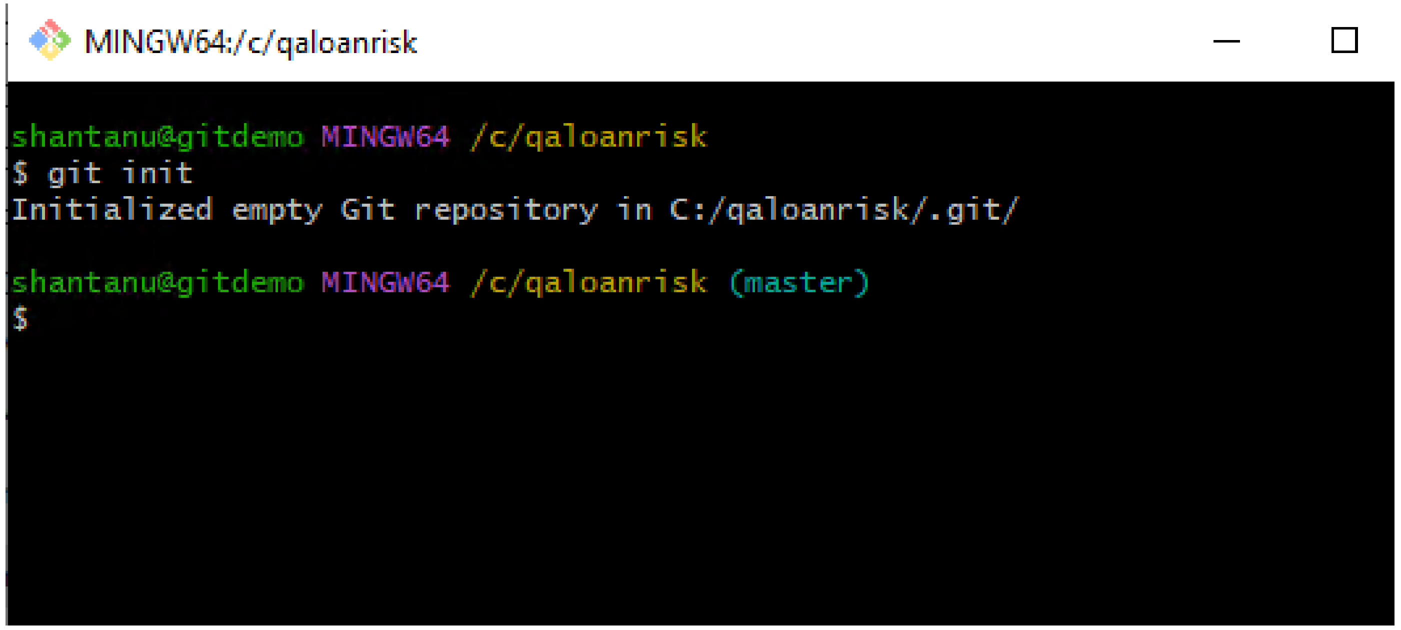


GIT INIT

git init is used to initialise the project directory

git init loanrisk

GIT INIT



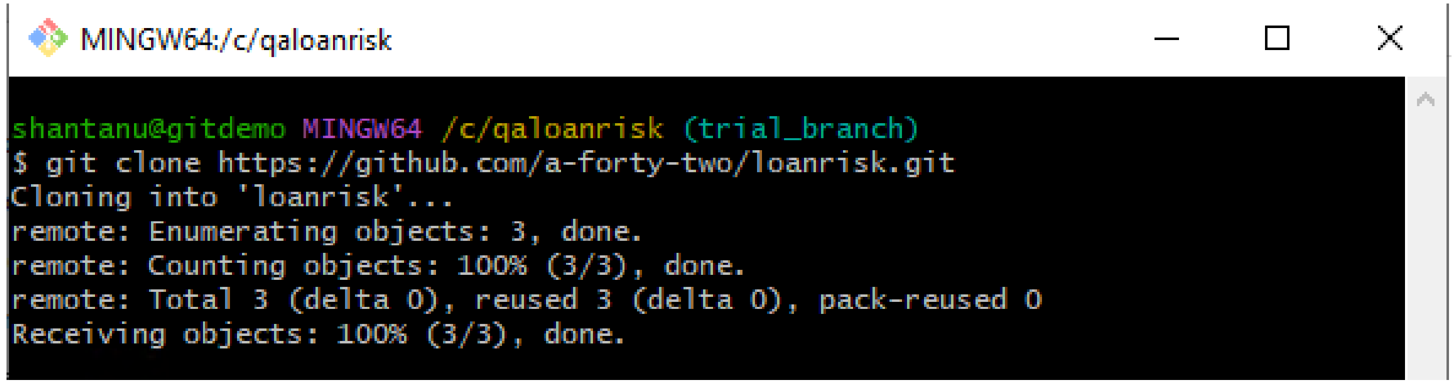
```
shantanu@gitdemo MINGW64 /c/qaloanrisk
$ git init
Initialized empty Git repository in C:/qaloanrisk/.git/

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$
```

GIT CLONE

git clone makes an identical copy of the latest version of a project in a repository and saves it to your computer.

```
git clone <https://name-of-the-repository-link>
```

GIT CLONE



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git clone https://github.com/a-forty-two/loanrisk.git
Cloning into 'loanrisk'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

Activity: (5 mins)
Create and Clone a repo

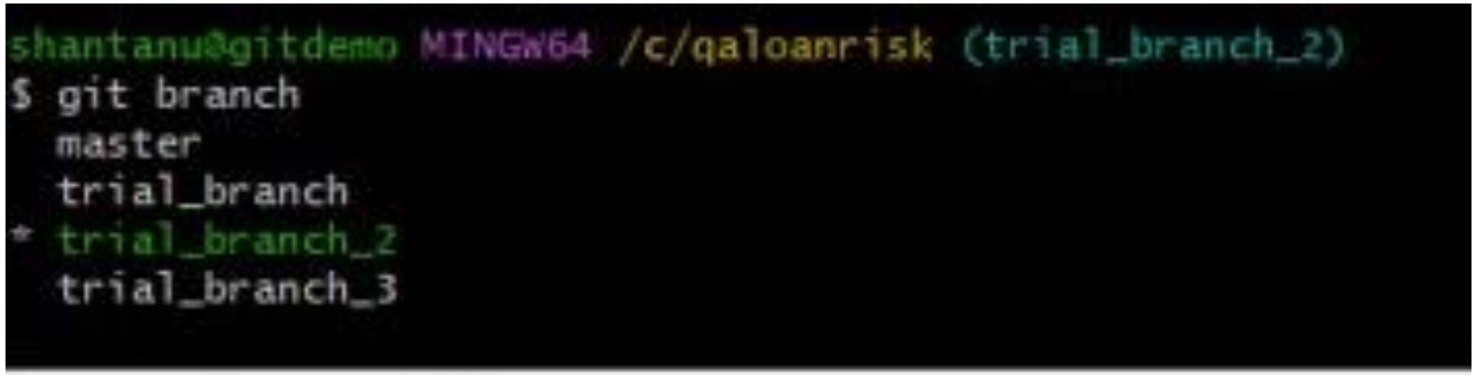
- Work individually, or in groups of 2 or 3.
- Manually upload a file in your repository on GitHub.com.
- Clone this repository in another folder on your drive.

GIT BRANCH

git branch command is used for creating, listing and deleting branches.

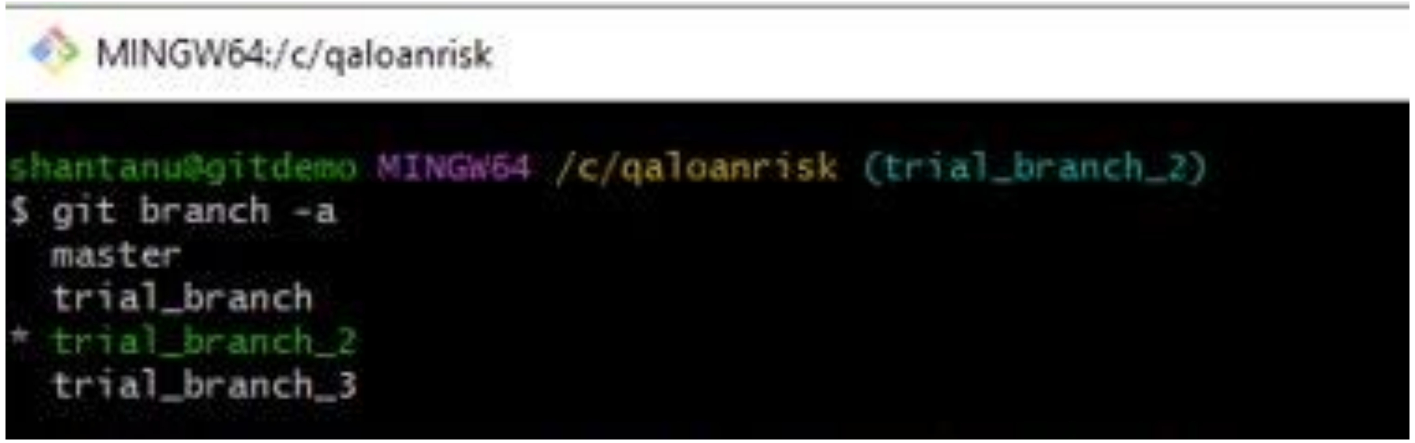
git branch <branch-name>

GIT BRANCH



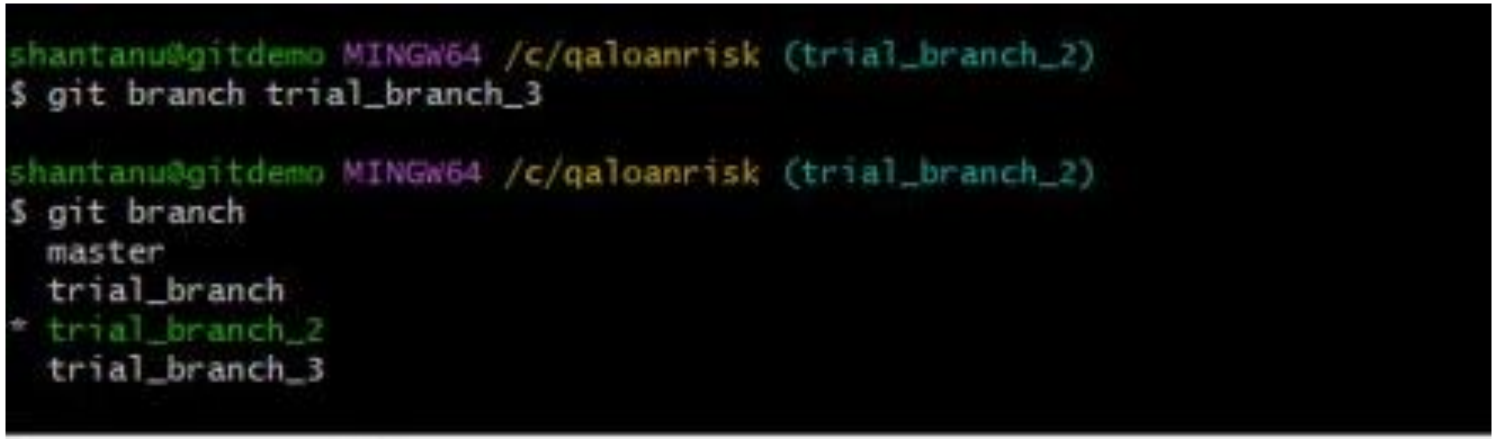
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git branch
  master
  trial_branch
* trial_branch_2
  trial_branch_3
```

GIT BRANCH


MINGW64:/c/qaloanrisk

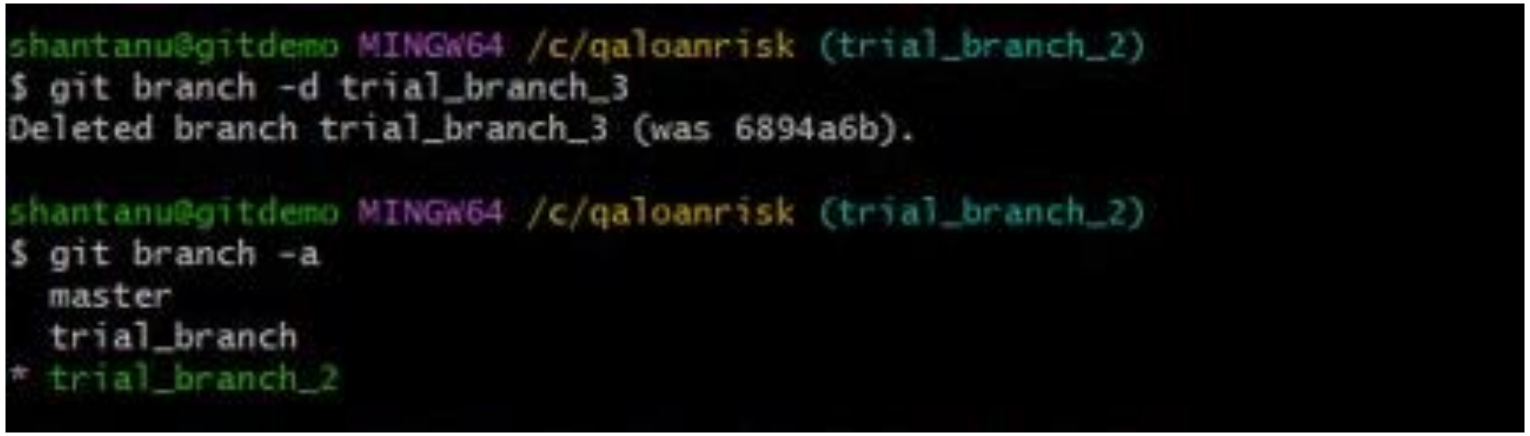
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git branch -a
  master
  trial_branch
* trial_branch_2
  trial_branch_3
```

GIT BRANCH



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git branch trial_branch_3

shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git branch
  master
  trial_branch
* trial_branch_2
  trial_branch_3
```

GIT BRANCH



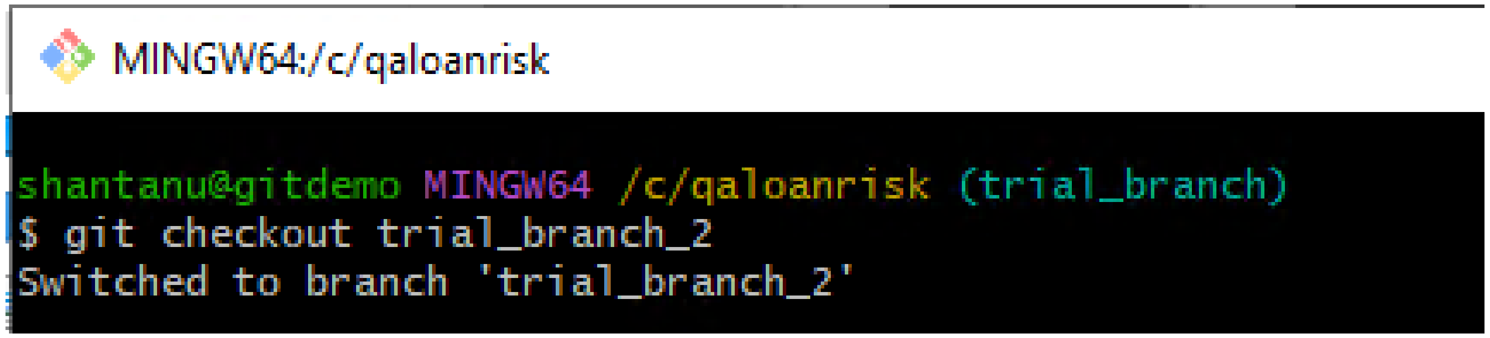
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git branch -d trial_branch_3
Deleted branch trial_branch_3 (was 6894a6b).

shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git branch -a
  master
  trial_branch
* trial_branch_2
```

GIT CHECKOUT

git checkout is used for switching from one branch to another.

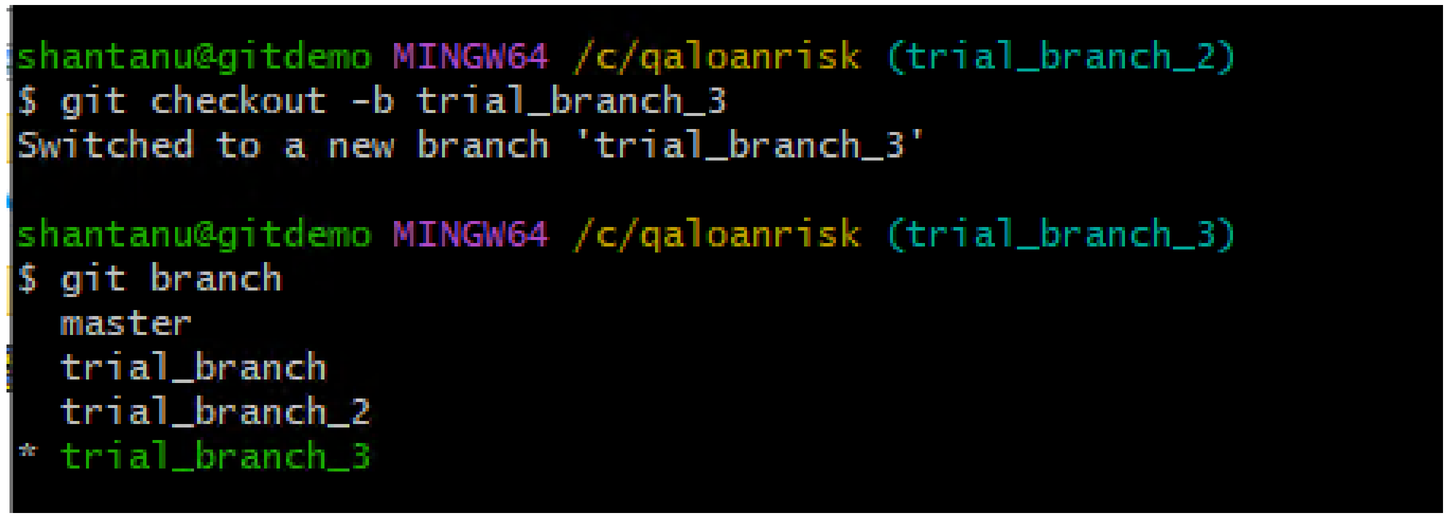
git checkout <name-of-your-branch>

GIT CHECKOUT


MINGW64:/c/qaloanrisk

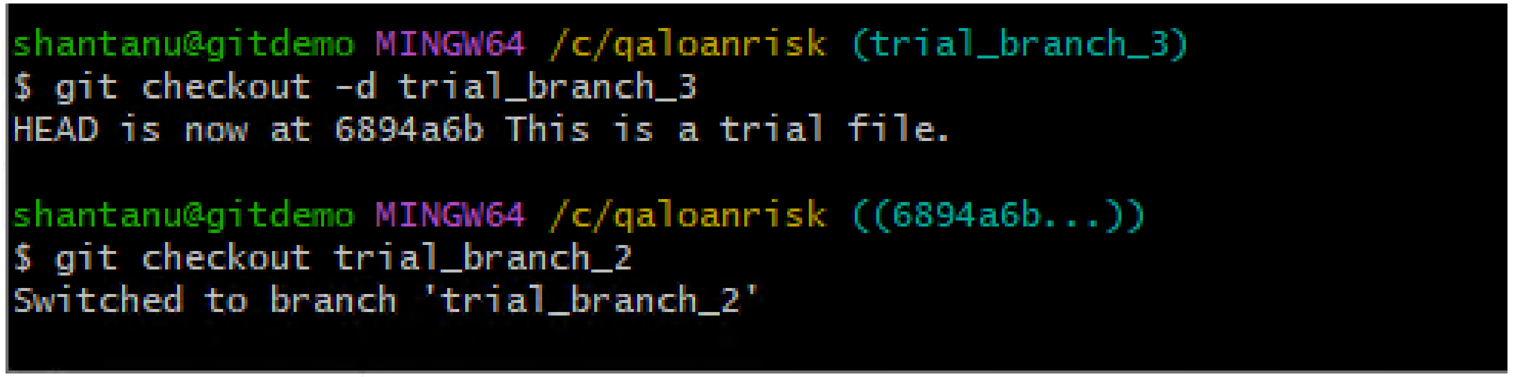
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git checkout trial_branch_2
Switched to branch 'trial_branch_2'
```

GIT CHECKOUT



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git checkout -b trial_branch_3
Switched to a new branch 'trial_branch_3'

shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_3)
$ git branch
  master
  trial_branch
  trial_branch_2
* trial_branch_3
```

GIT CHECKOUT



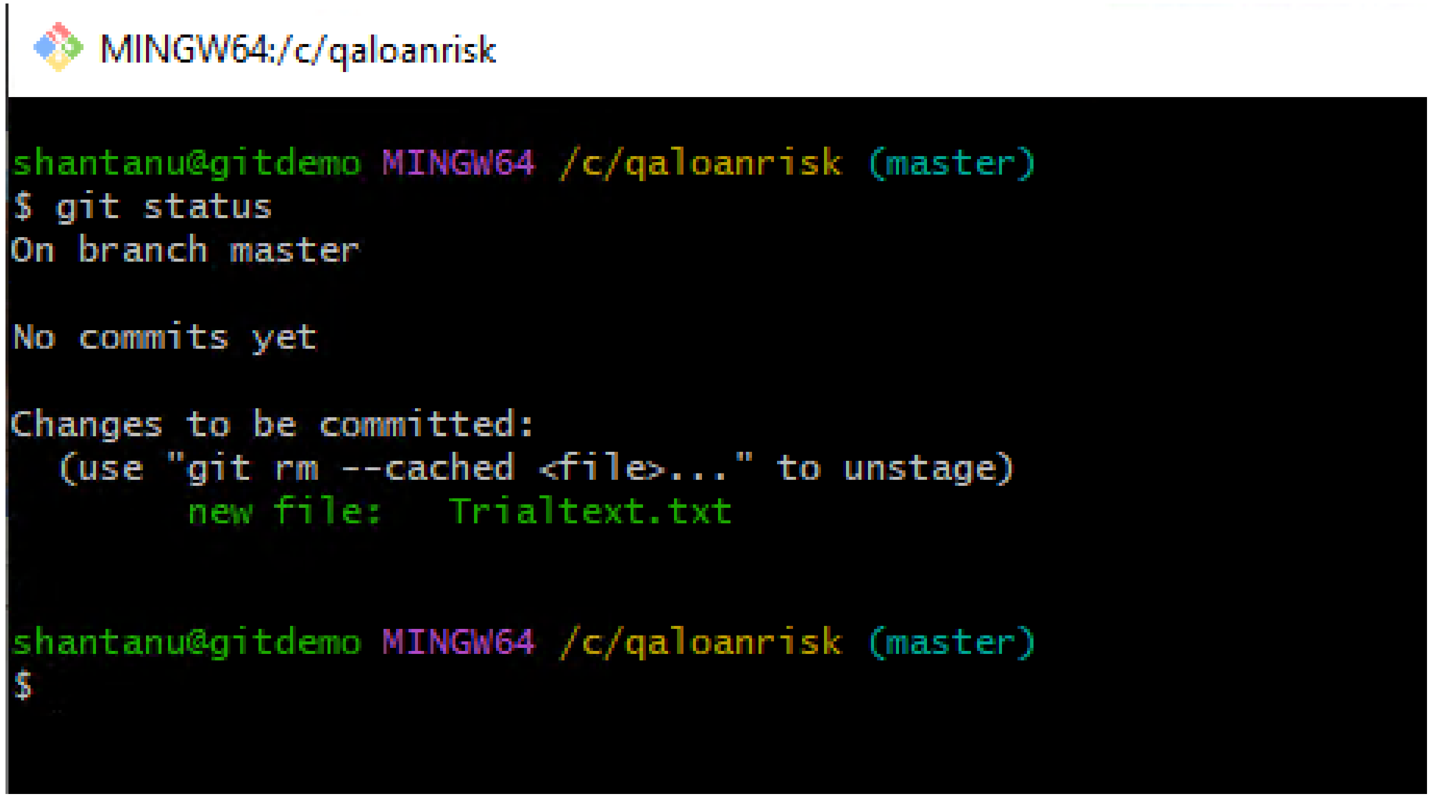
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_3)
$ git checkout -d trial_branch_3
HEAD is now at 6894a6b This is a trial file.

shantanu@gitdemo MINGW64 /c/qaloanrisk ((6894a6b...))
$ git checkout trial_branch_2
Switched to branch 'trial_branch_2'
```

GIT STATUS

The ***git status*** command gives us all the information about the current branch.

git status

GIT STATUS


MINGW64:/c/qaloanrisk

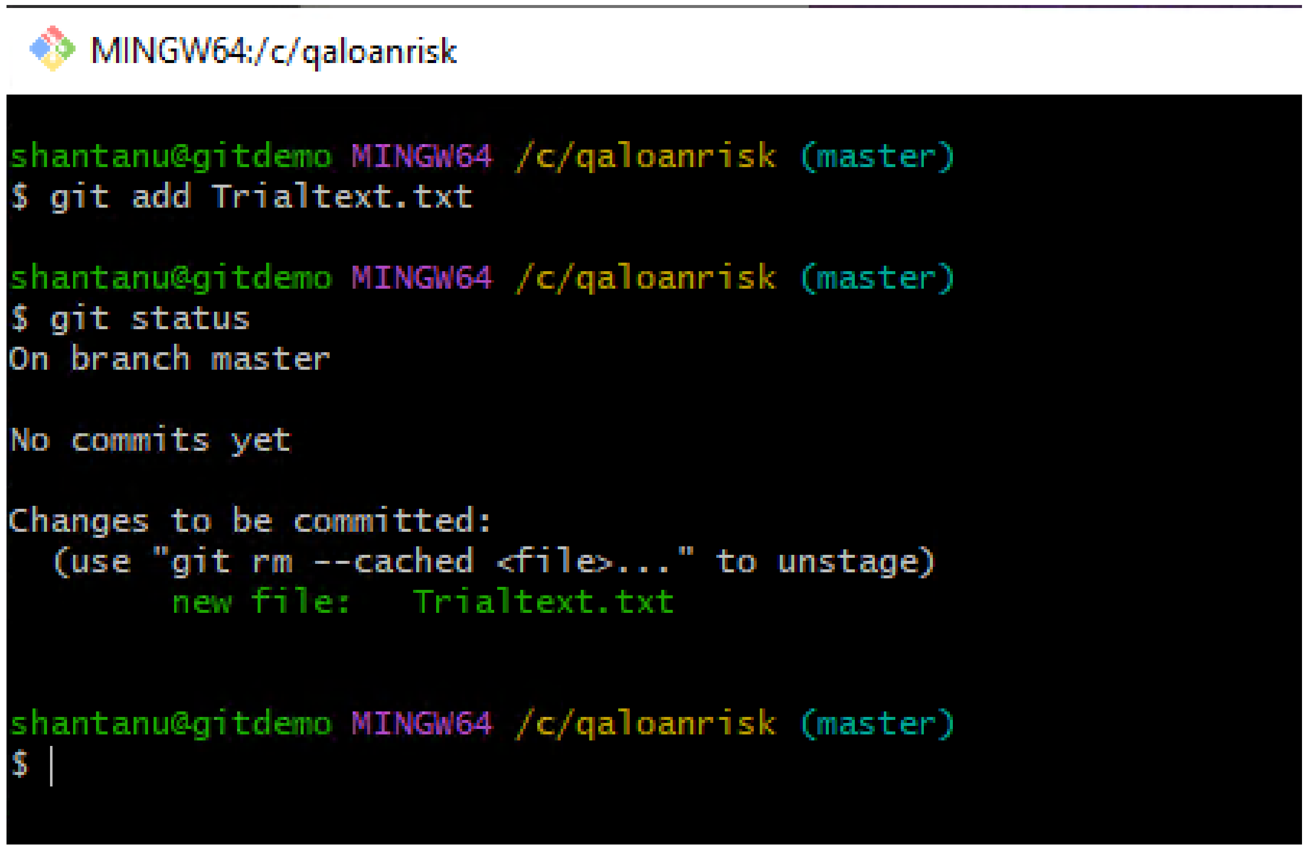
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Trialtext.txt


shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$
```

GIT ADD

git add command is used to include the changes of a file into the next commit.

```
git add <file>
```

GIT ADD


MINGW64:/c/qaloanrisk

```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git add Trialtext.txt

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Trialtext.txt


shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ |
```

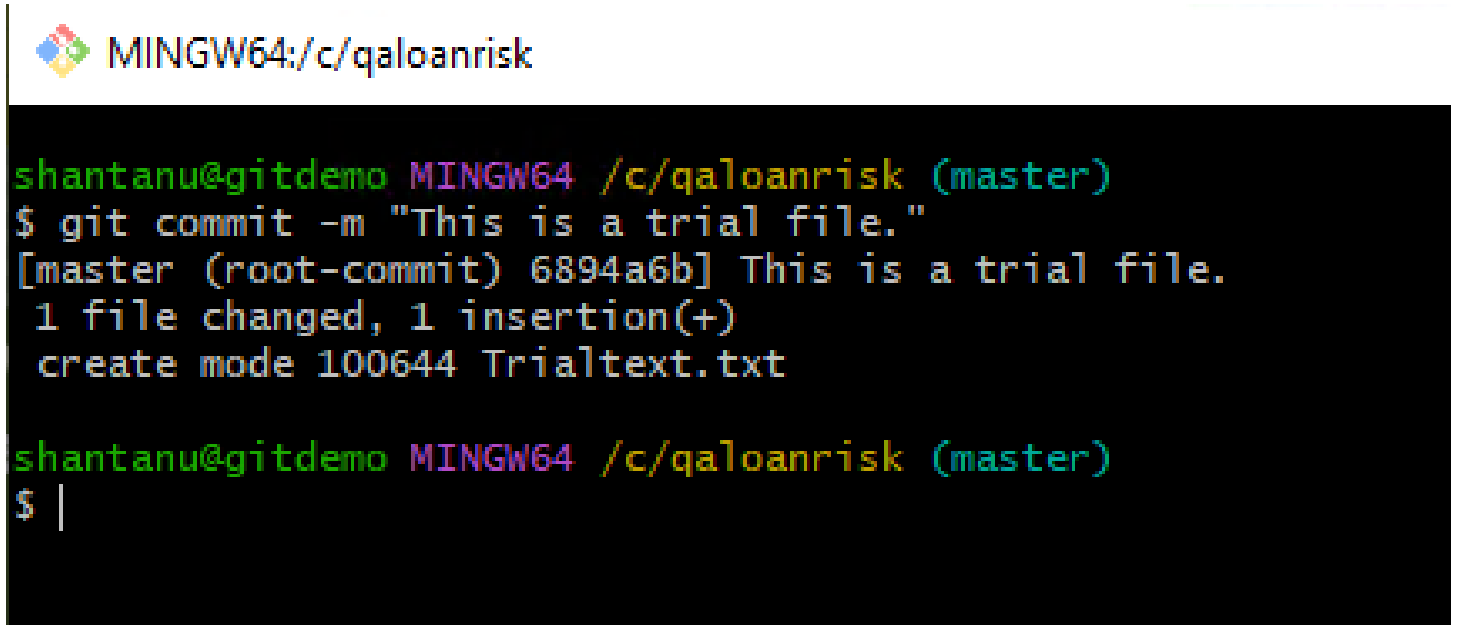
GIT COMMIT

git commit sets a checkpoint in the development process which you can go back to later when needed.

It is required to write a short message to explain what we have developed or changed in the source code.

git commit -m "commit message"

GIT COMMIT


MINGW64:/c/qaloanrisk

```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git commit -m "This is a trial file."
[master (root-commit) 6894a6b] This is a trial file.
 1 file changed, 1 insertion(+)
 create mode 100644 Trialtext.txt

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ 
```

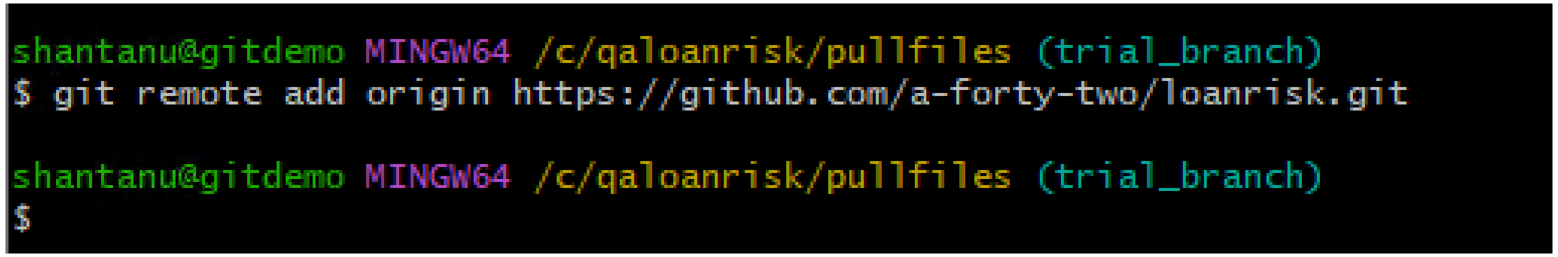
REMOTE ORIGIN

git remote origin

After the local repository is validated, the next step is to create a remote repository that the local repository will connect to.

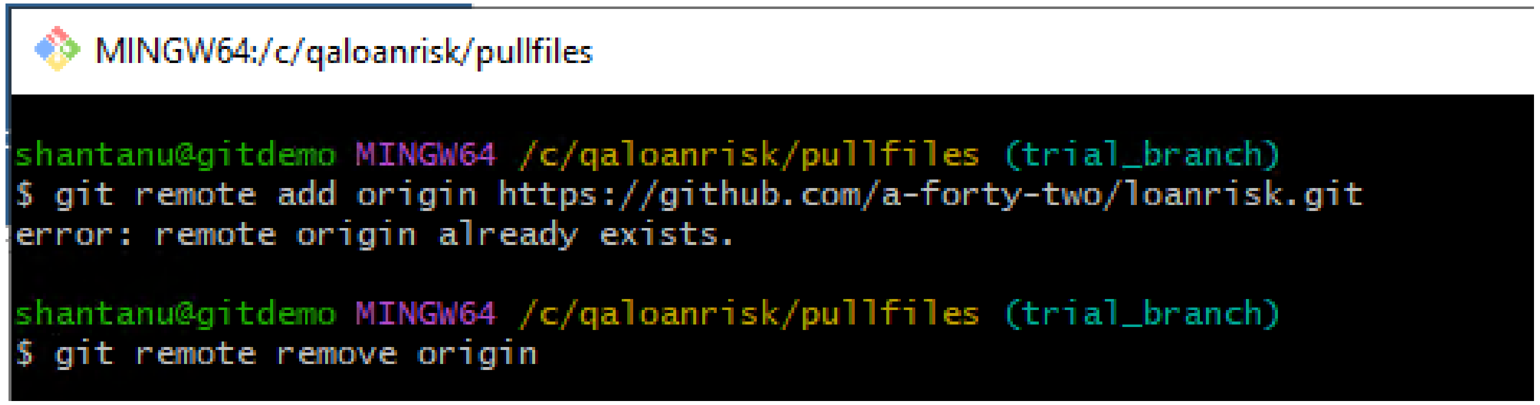
```
git remote add origin <https link of your repository>
```

REMOTE ORIGIN



```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$ git remote add origin https://github.com/a-forty-two/loanrisk.git

shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$
```

REMOTE ORIGIN



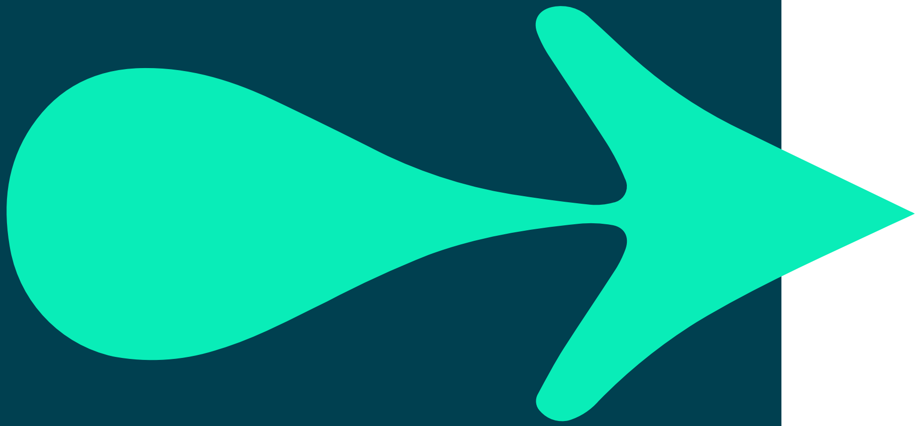
MINGW64:/c/qaloanrisk/pullfiles

```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$ git remote add origin https://github.com/a-forty-two/loanrisk.git
error: remote origin already exists.

shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$ git remote remove origin
```

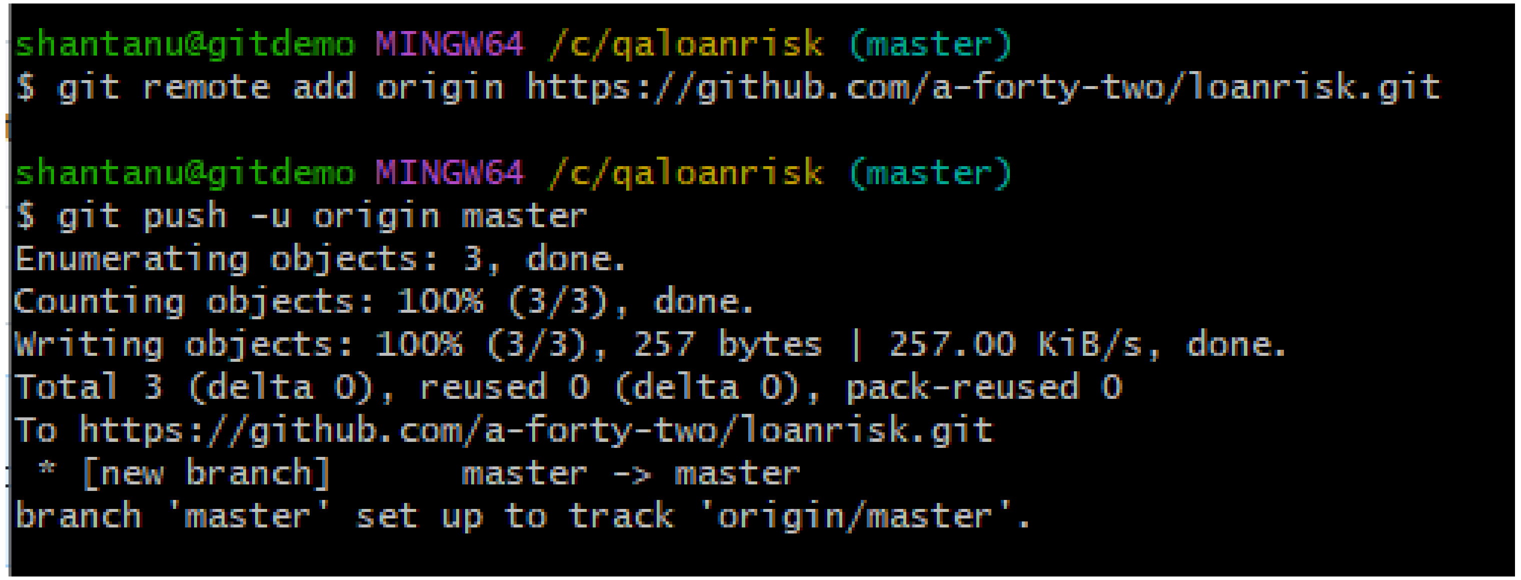
GIT PUSH

git push uploads your commits to the remote repository.

git push <remote> <branch-name>



GIT PUSH



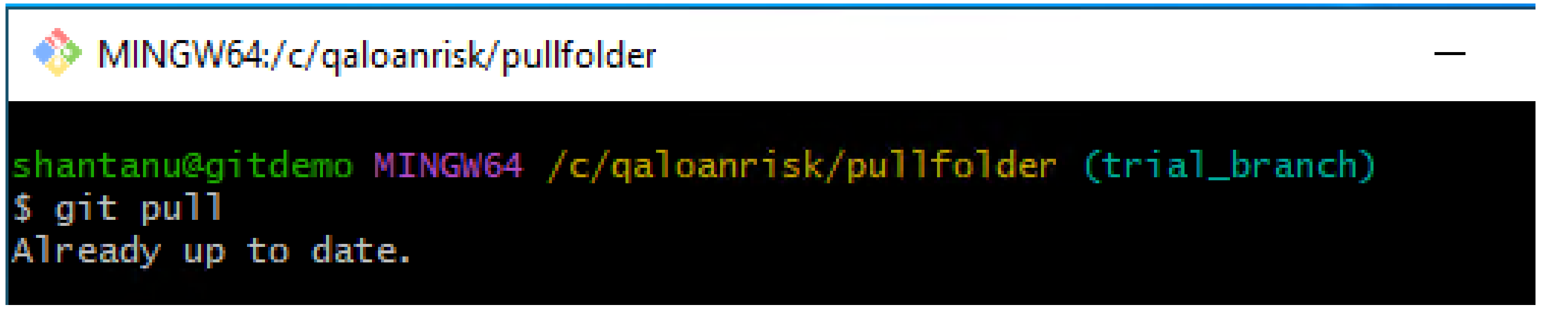
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git remote add origin https://github.com/a-forty-two/loanrisk.git

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 257 bytes | 257.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/a-forty-two/loanrisk.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

GIT PULL

The ***git pull*** command is used to get updates from the remote repo.

git pull <remote>

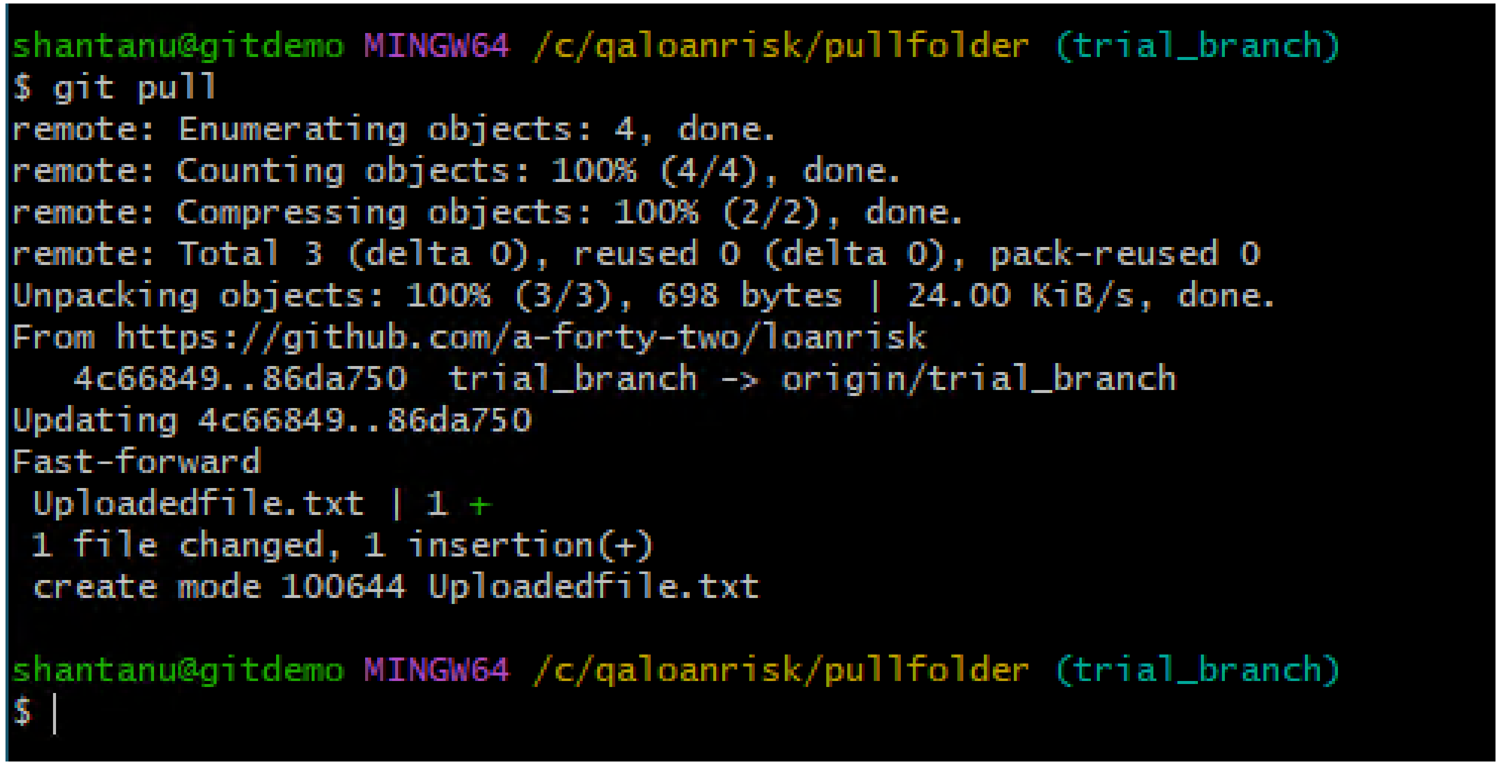
GIT PULL



MINGW64:/c/qaloanrisk/pullfolder

```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfolder (trial_branch)
$ git pull
Already up to date.
```

GIT PULL



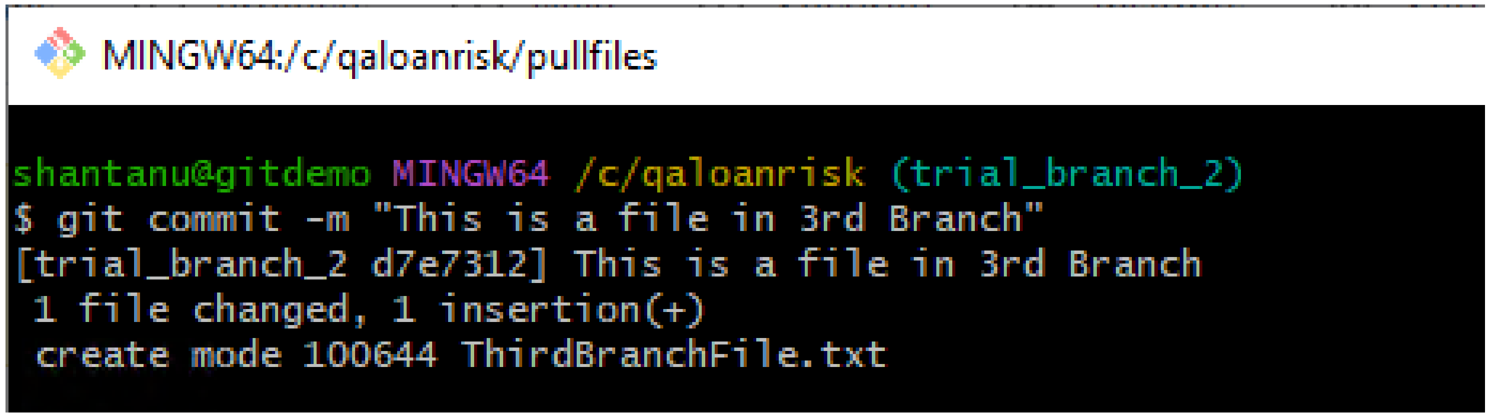
```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfolder (trial_branch)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 698 bytes | 24.00 KiB/s, done.
From https://github.com/a-forty-two/loanrisk
   4c66849..86da750  trial_branch -> origin/trial_branch
Updating 4c66849..86da750
Fast-forward
 Uploadedfile.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Uploadedfile.txt

shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfolder (trial_branch)
$ |
```

GIT REVERT

git revert command is used to undo our commits or changes either locally or remotely. We must also mention the hash code next to the commit which we wish to undo.

git revert 3321844

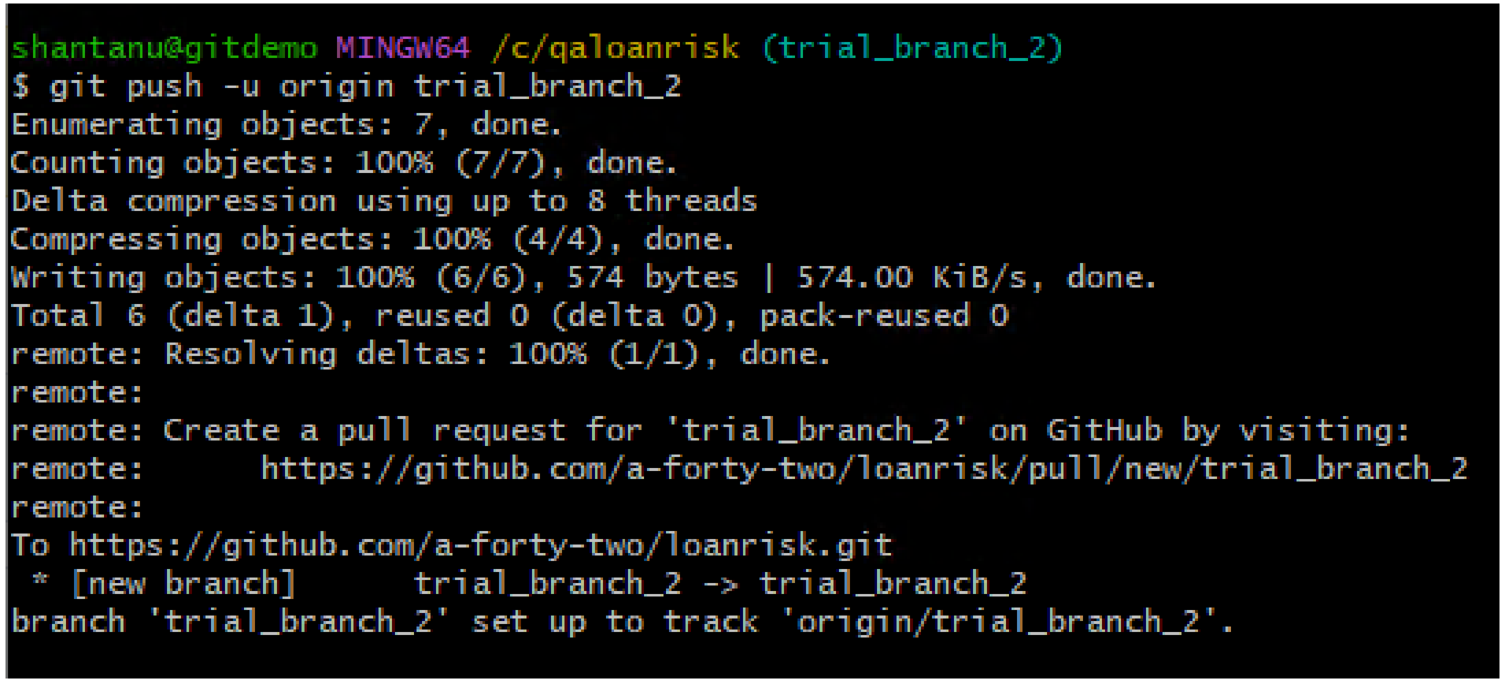
GIT REVERT



MINGW64:/c/qaloanrisk/pullfiles

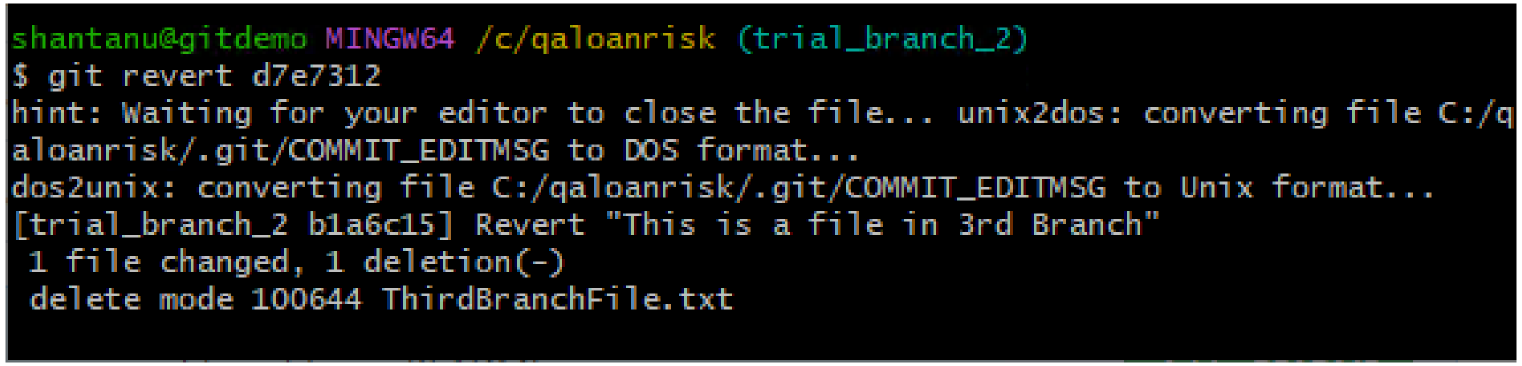
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git commit -m "This is a file in 3rd Branch"
[trial_branch_2 d7e7312] This is a file in 3rd Branch
 1 file changed, 1 insertion(+)
 create mode 100644 ThirdBranchFile.txt
```

GIT REVERT



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git push -u origin trial_branch_2
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 574 bytes | 574.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'trial_branch_2' on GitHub by visiting:
remote:      https://github.com/a-forty-two/loanrisk/pull/new/trial_branch_2
remote:
To https://github.com/a-forty-two/loanrisk.git
 * [new branch]      trial_branch_2 -> trial_branch_2
branch 'trial_branch_2' set up to track 'origin/trial_branch_2'.
```

GIT REVERT



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git revert d7e7312
hint: Waiting for your editor to close the file... unix2dos: converting file C:/q
aloanrisk/.git/COMMIT_EDITMSG to DOS format...
dos2unix: converting file C:/qaloanrisk/.git/COMMIT_EDITMSG to Unix format...
[trial_branch_2 b1a6c15] Revert "This is a file in 3rd Branch"
 1 file changed, 1 deletion(-)
 delete mode 100644 ThirdBranchFile.txt
```

GIT MERGE

git merge command is used to integrate or merge the feature branch with the parent branch.

git merge <branch-name>

Activity: (30 mins)
Make changes to a repo

- Work individually, or in groups of 2 or 3.
- Create a new branch for your repository.
- Using the commands that you have learnt, add a text file containing "Hello QA!”
- Make changes to the file and add "Bye bye!" towards the end.
- Commit the changes and revert them.
- Merge the branch with master.
- Pull the changes made in the master to the local repo.
- Push all together to a new branch called Finished Product.

GIT FLOW

- Git flow makes use of the feature branches and multiple primary branches.
- It was published by Vincent Driessen at nvie.
- Git flow simplifies the process of parallel development as it seperates the new development from the released version.
- Git flow provides a powerful framework for managing huge projects.
- It is best suited for projects following a scheduled release cycle and for continuous delivery.

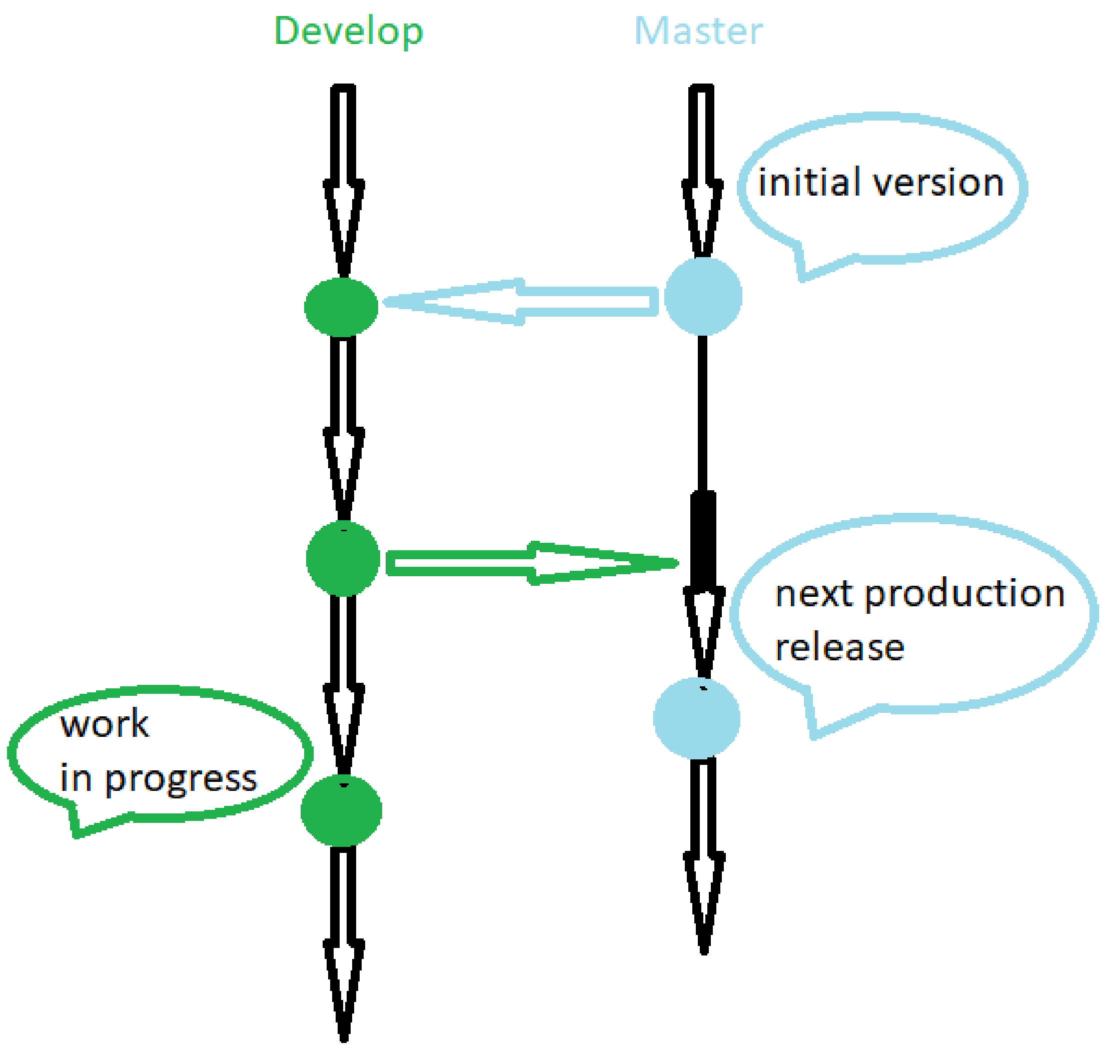
WHAT IS A BRANCH?

- Git branches are a pointer to a record of your modifications. If you want to add a new feature or resolve an issue, you create a new branch to contain your changes.

- Branching is used in version control to preserve stability while making small-scale changes to the code.

- Branching simplifies the process of bug fixing, adding new features, and integrating new versions after they have been tested in isolation.

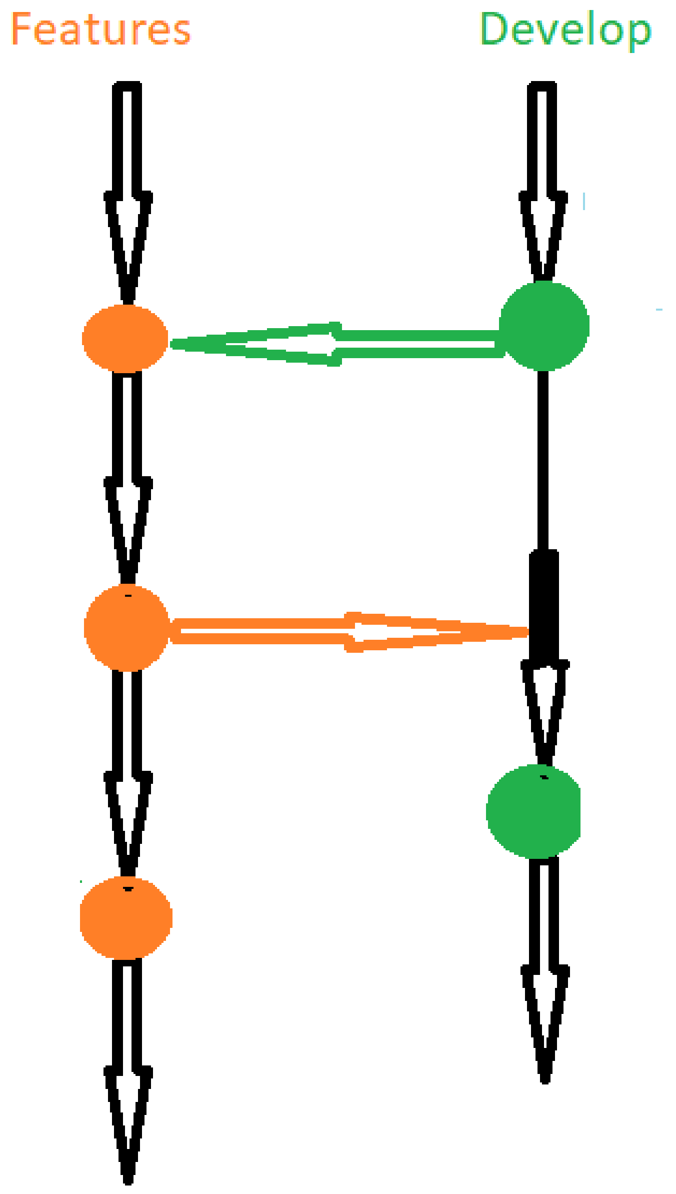
GIT FLOW


Develop
Master
initial version
next production release
work in progress

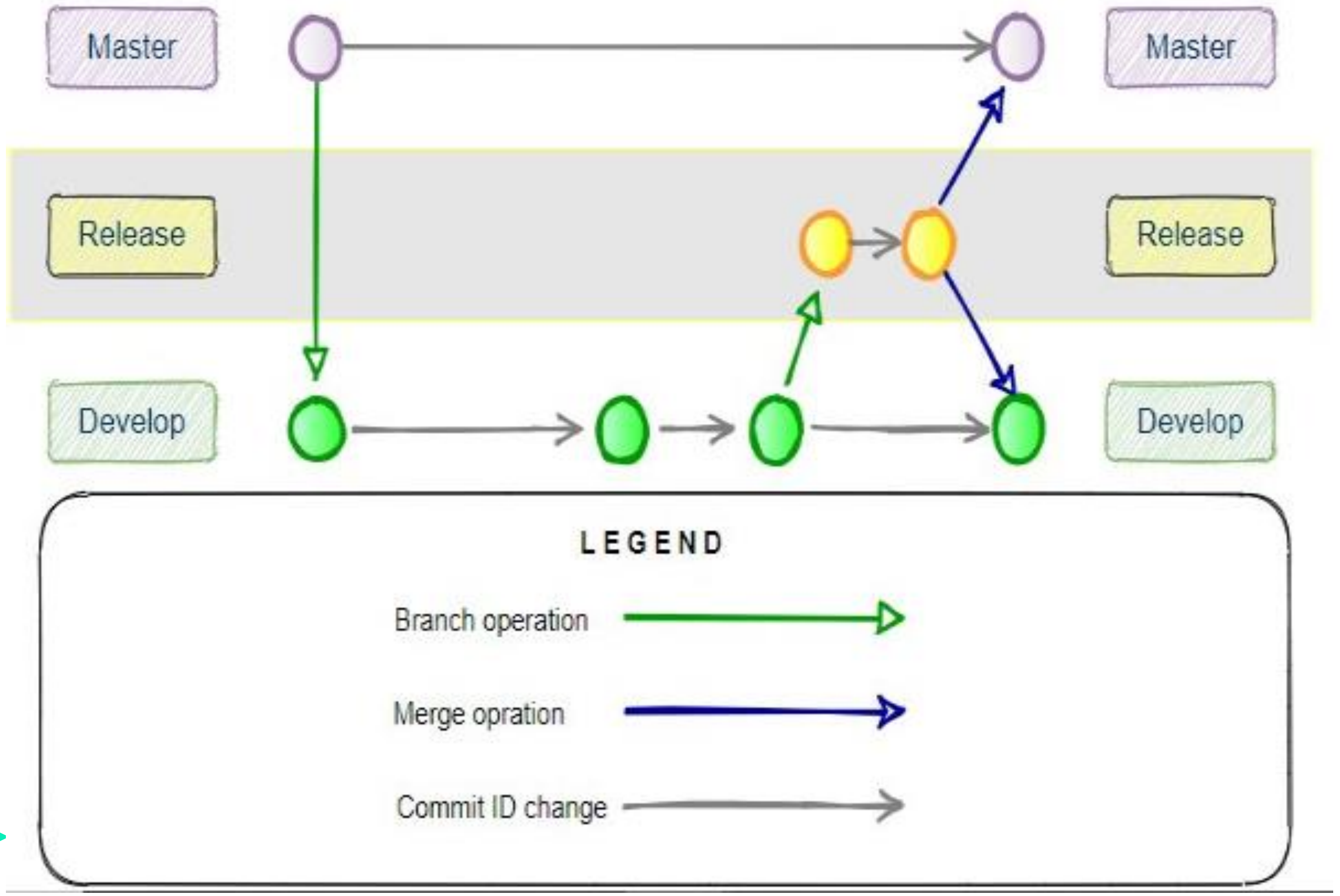
FEATURE BRANCHES

- The key aspect of Feature Branch is that all feature development ought to happen in a separate branch and not the main branch.
- Multiple developers can easily work on a specific feature using this encapsulation without interfering with the main codebase.

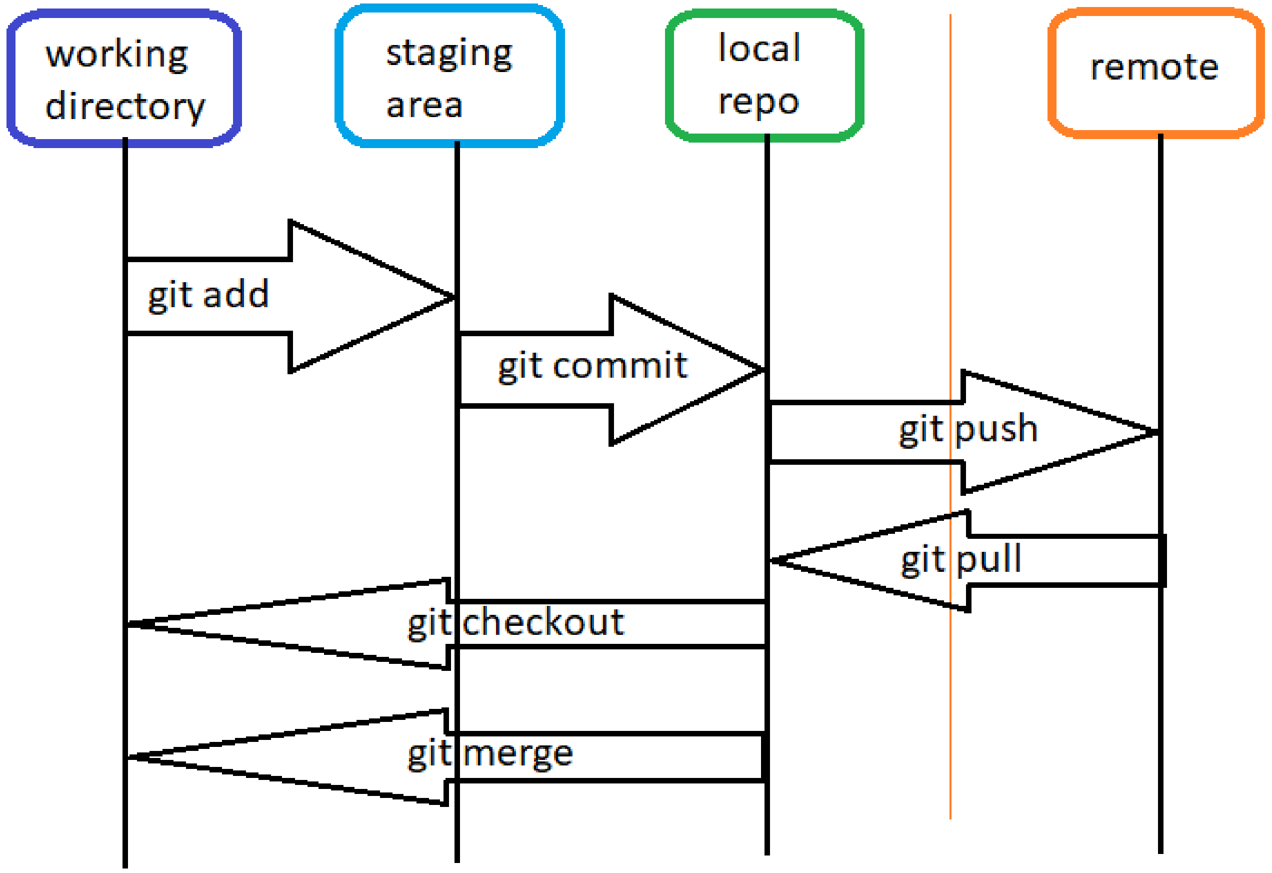
GIT FLOW



GIT FLOW


Master
Master
Release
Release
Develop
Develop
LEGEND
Branch operation
Merge opration
Commit ID change

GIT FLOW


working directory
staging area
local repo
remote
git add
git commit
git push
git pull
git checkout
git merge

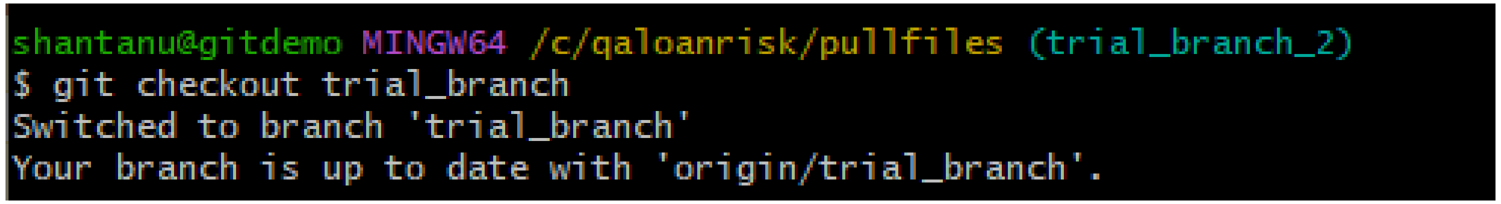
PULL REQUESTS

- A developer can use pull requests to let the rest of the team know when a feature is finished.
- This notifies all concerned parties that the code needs to be reviewed and merged into the main branch.
- It also acts as a forum to discuss the proposed features.
- Team members can provide comments in the pull request and even modify the functionality by submitting further commits if there are any issues with the modifications.
- The pull request contains a direct tracking system for all the activities.

MERGE REQUESTS

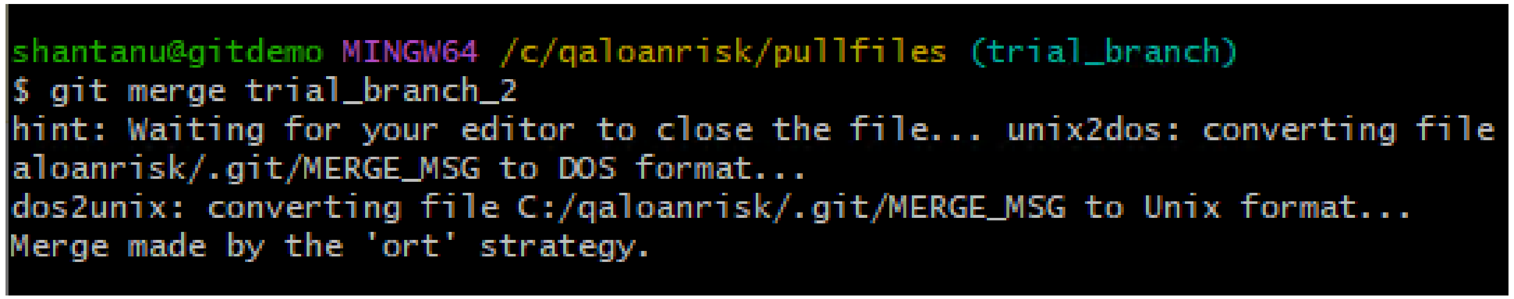
- If you are working in a feature branch and wish to merge your change in the main branch, you must make a merge request (eg. Master branch).
- Using merge requests as a code review mechanism, anyone (often other developers) can commit and push a fix if your code reveals flaws or problems.
- Using merge requests, you may simply share the modifications you've made to a project with other individuals and interchange the code.

HOW TO MERGE?



```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch_2)
$ git checkout trial_branch
Switched to branch 'trial_branch'
Your branch is up to date with 'origin/trial_branch'.
```

HOW TO MERGE?



```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$ git merge trial_branch_2
hint: Waiting for your editor to close the file... unix2dos: converting file
aloanrisk/.git/MERGE_MSG to DOS format...
dos2unix: converting file C:/qaloanrisk/.git/MERGE_MSG to Unix format...
Merge made by the 'ort' strategy.
```

RESOLVING MERGE CONFLICTS

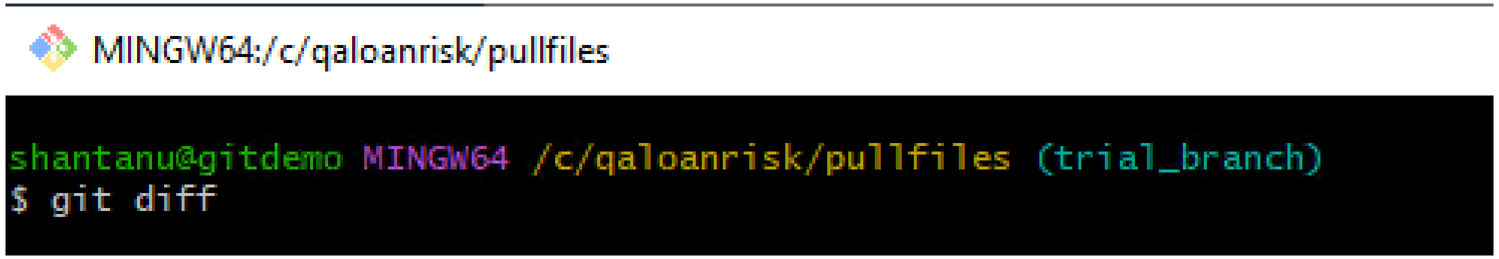
- When Git is unable to automatically reconcile discrepancies in code between two commits, an event known as a merge conflict occurs.
- Only if the commits are on distinct lines or branches can Git automatically merge the changes.
- Combining multiple branches and resolving any conflicting edits are the primary responsibilities of the git merge command.

GIT DIFF

git diff shows the difference between files, it could be between files in different staging, or different commits, or it shows the differences between the two branches mentioned.

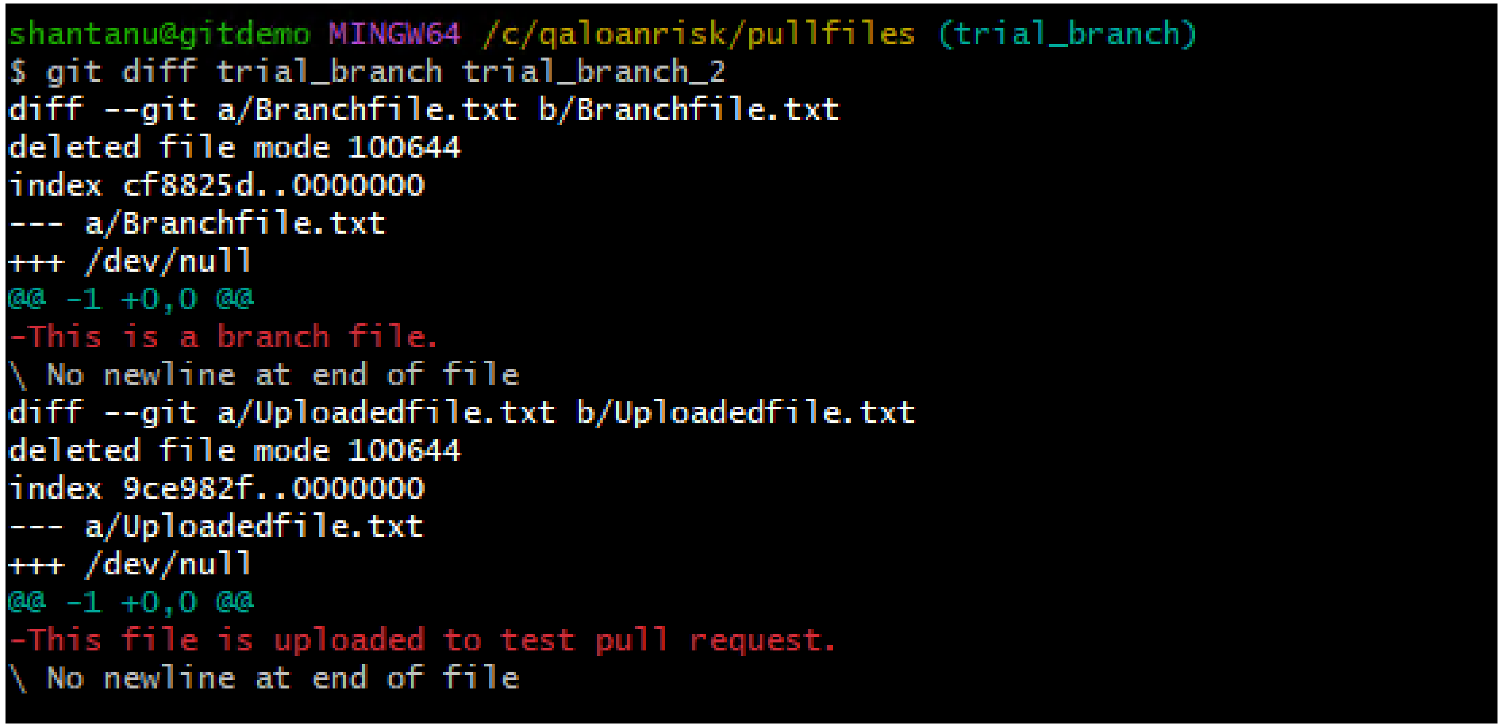
git diff [first branch] [second branch]

GIT DIFF


MINGW64:/c/qaloanrisk/pullfiles

```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$ git diff
```

GIT DIFF



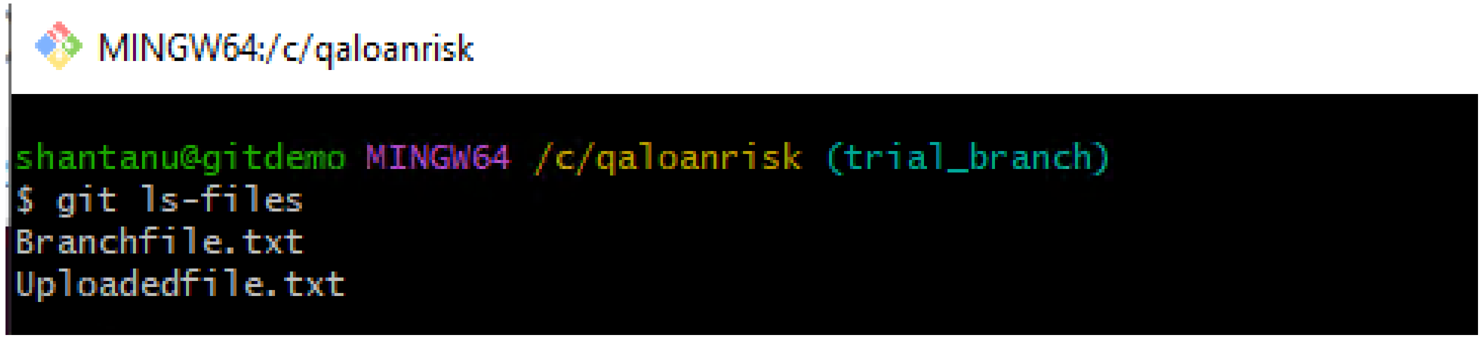
```
shantanu@gitdemo MINGW64 /c/qaloanrisk/pullfiles (trial_branch)
$ git diff trial_branch trial_branch_2
diff --git a/Branchfile.txt b/Branchfile.txt
deleted file mode 100644
index cf8825d..0000000
--- a/Branchfile.txt
+++ /dev/null
@@ -1 +0,0 @@
-This is a branch file.
\ No newline at end of file
diff --git a/Uploadedfile.txt b/Uploadedfile.txt
deleted file mode 100644
index 9ce982f..0000000
--- a/Uploadedfile.txt
+++ /dev/null
@@ -1 +0,0 @@
-This file is uploaded to test pull request.
\ No newline at end of file
```

GIT LS-FILES

git ls-files Show information about files in the index and the working tree.

```
git ls-files
```

GIT LS-FILES



MINGW64:/c/qaloanrisk

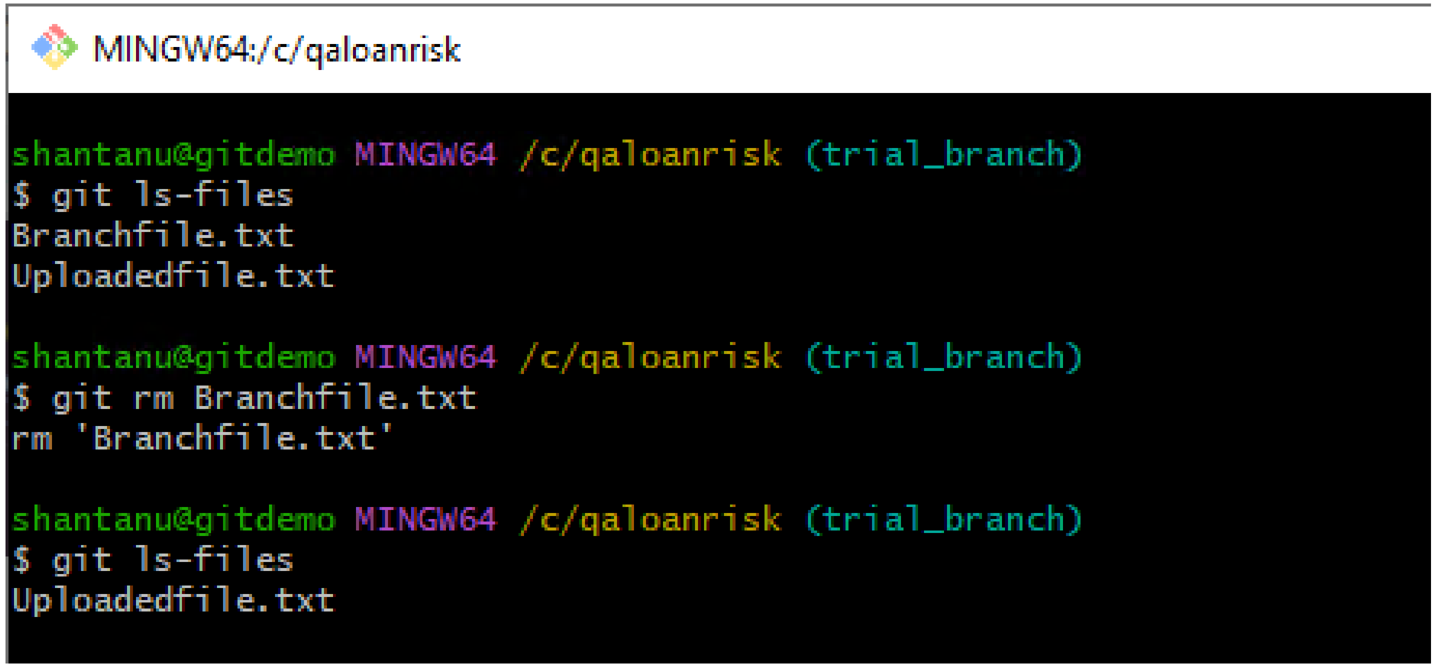
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git ls-files
Branchfile.txt
Uploadedfile.txt
```

GIT RM

git rm This command deletes the file from your working directory and stages the deletion.

```
git rm <filename>
```

GIT RM


MINGW64:/c/qaloanrisk

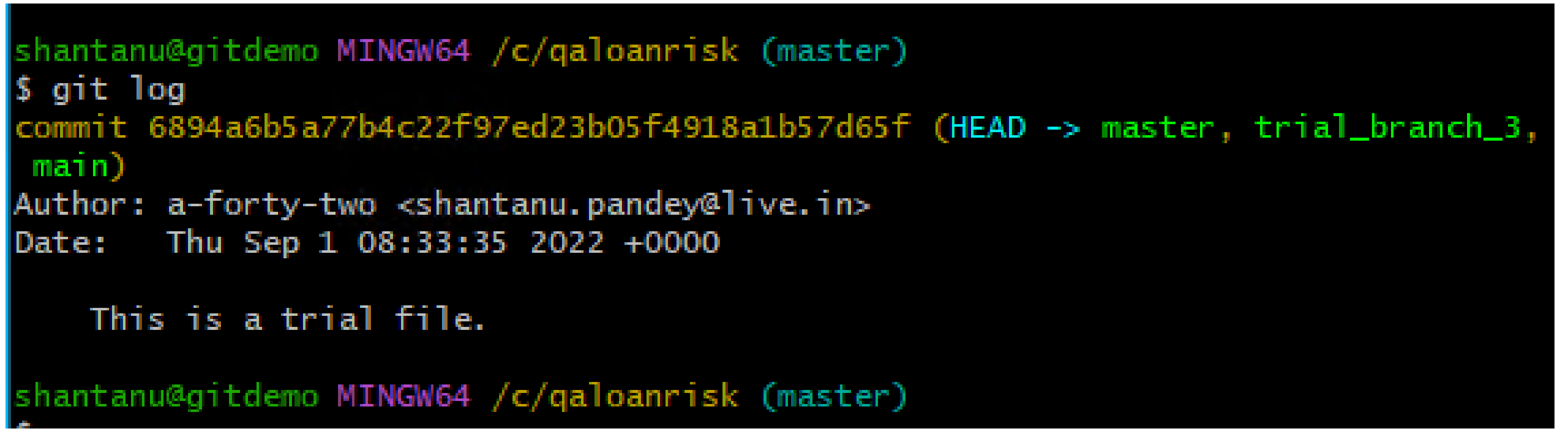
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git ls-files
Branchfile.txt
Uploadedfile.txt

shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git rm Branchfile.txt
rm 'Branchfile.txt'

shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git ls-files
Uploadedfile.txt
```

GIT LOG

git log This command is used to list the version history for the current branch.

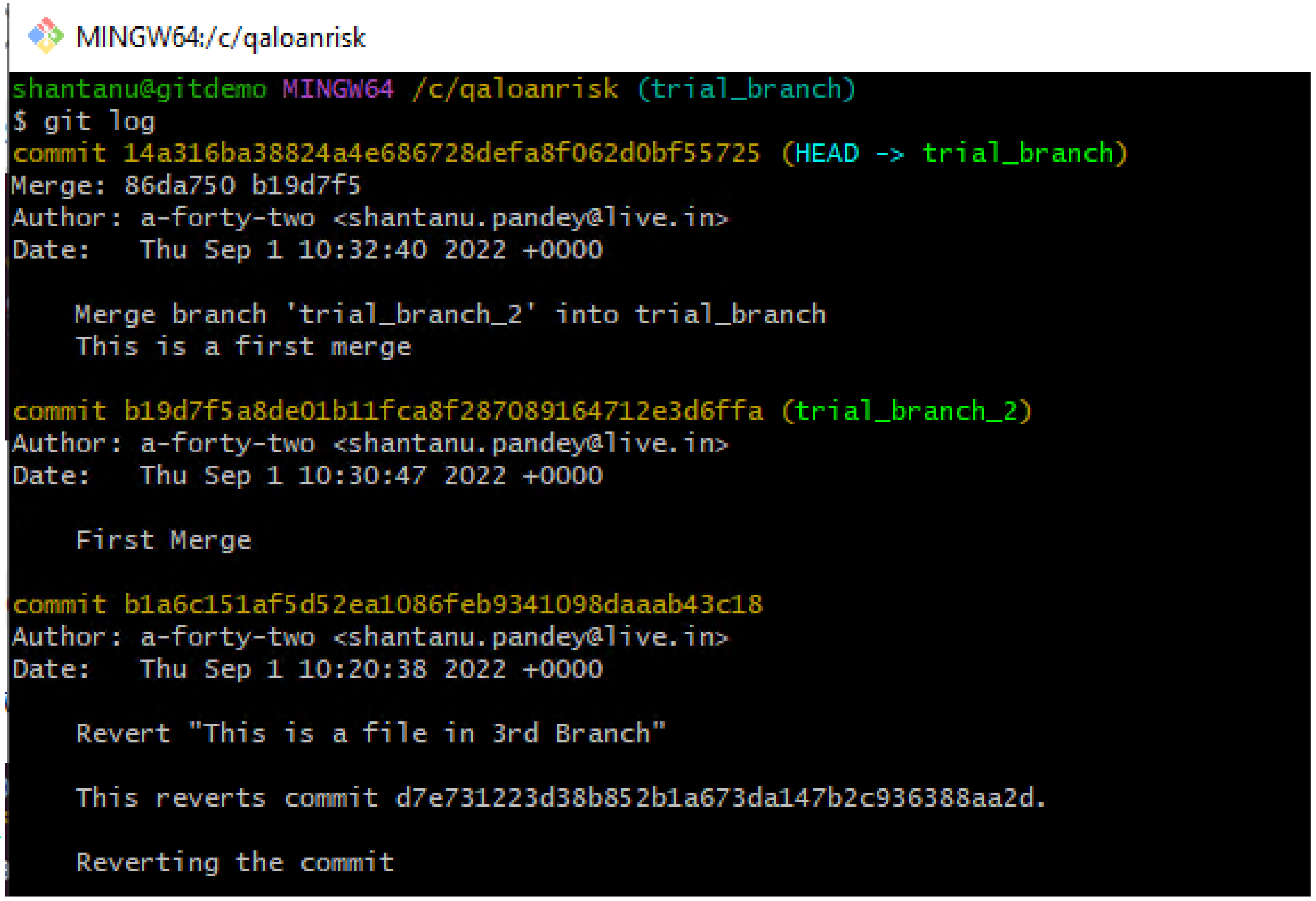
```
git log
```

GIT LOG



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git log
commit 6894a6b5a77b4c22f97ed23b05f4918a1b57d65f (HEAD -> master, trial_branch_3,
 main)
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 08:33:35 2022 +0000

    This is a trial file.

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
```

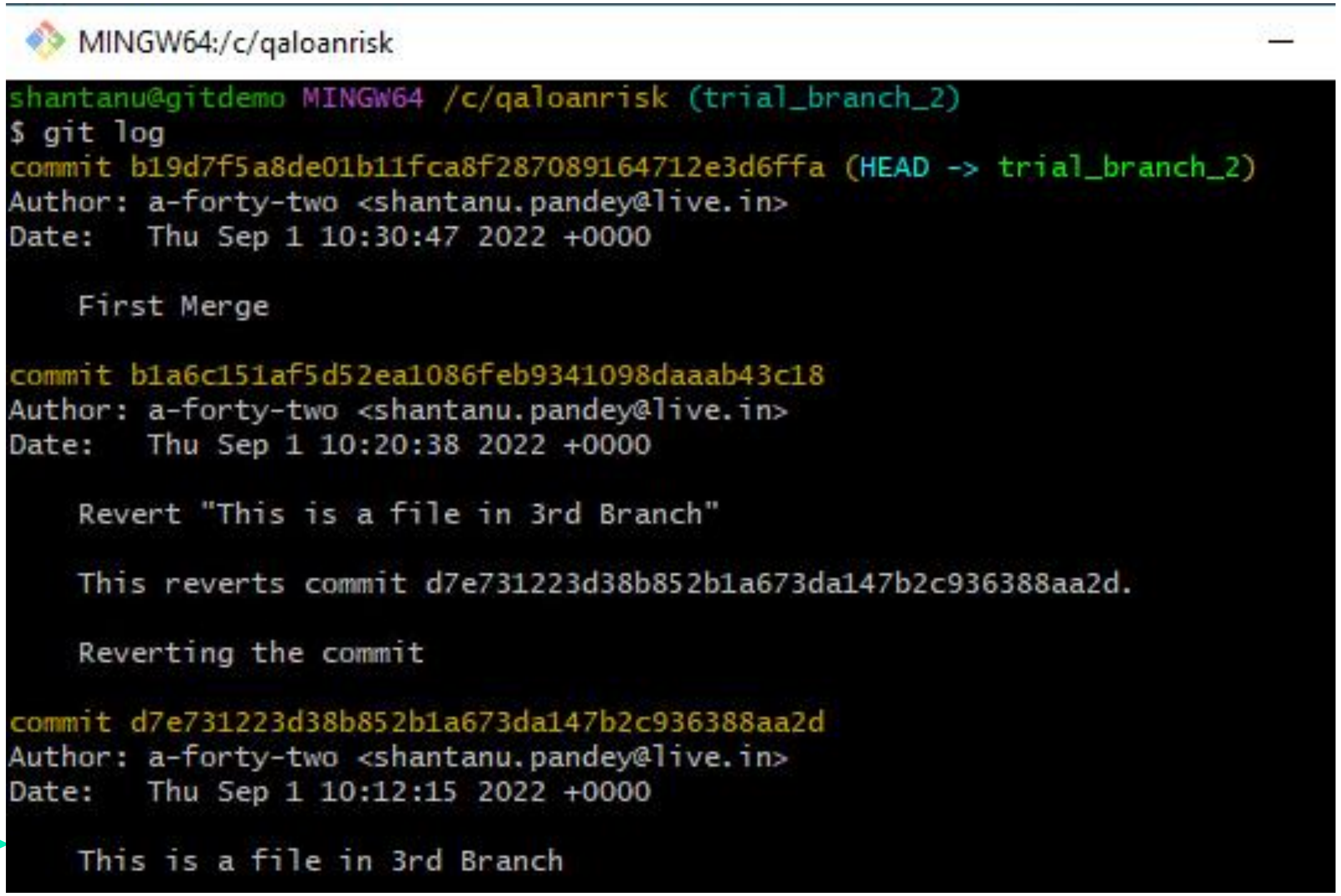
GIT LOG



MINGW64:/c/qaloanrisk

```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git log
commit 14a316ba38824a4e686728defa8f062d0bf55725 (HEAD -> trial_branch)
Merge: 86da750 b19d7f5
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:32:40 2022 +0000

    Merge branch 'trial_branch_2' into trial_branch
    This is a first merge

commit b19d7f5a8de01b11fca8f287089164712e3d6ffa (trial_branch_2)
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:30:47 2022 +0000

    First Merge

commit b1a6c151af5d52ea1086feb9341098daaab43c18
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:20:38 2022 +0000

    Revert "This is a file in 3rd Branch"

    This reverts commit d7e731223d38b852b1a673da147b2c936388aa2d.

    Reverting the commit
```

GIT LOG


MINGW64:/c/qaloanrisk

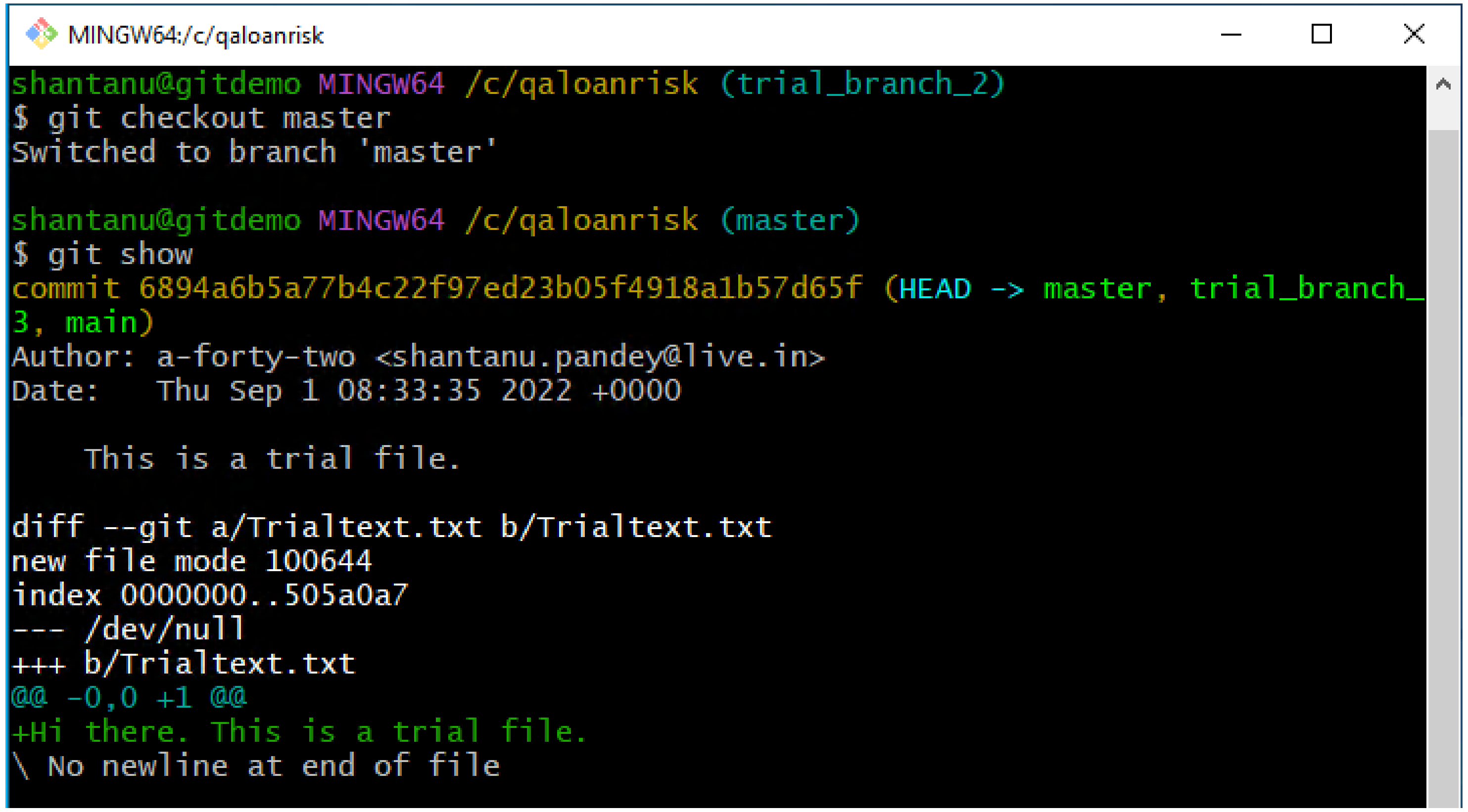
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git log
commit b19d7f5a8de01b11fca8f287089164712e3d6ffa (HEAD -> trial_branch_2)
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:30:47 2022 +0000

    First Merge

commit b1a6c151af5d52ea1086feb9341098daaab43c18
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:20:38 2022 +0000

    Revert "This is a file in 3rd Branch"

    This reverts commit d7e731223d38b852b1a673da147b2c936388aa2d.

    Reverting the commit

commit d7e731223d38b852b1a673da147b2c936388aa2d
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:12:15 2022 +0000

    This is a file in 3rd Branch
```

GIT SHOW

git show this command shows the metadata and content changes of the specified commit.

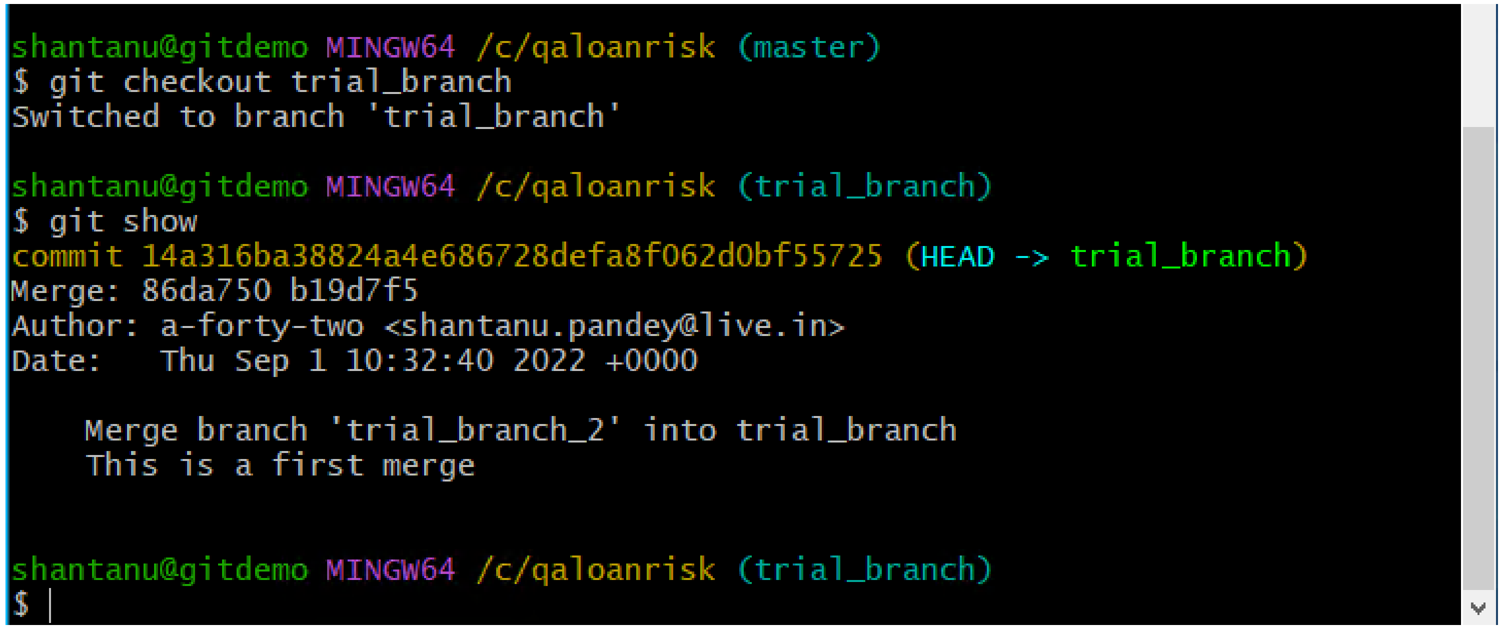
```
git show
```

GIT SHOW


MINGW64:/c/qaloanrisk

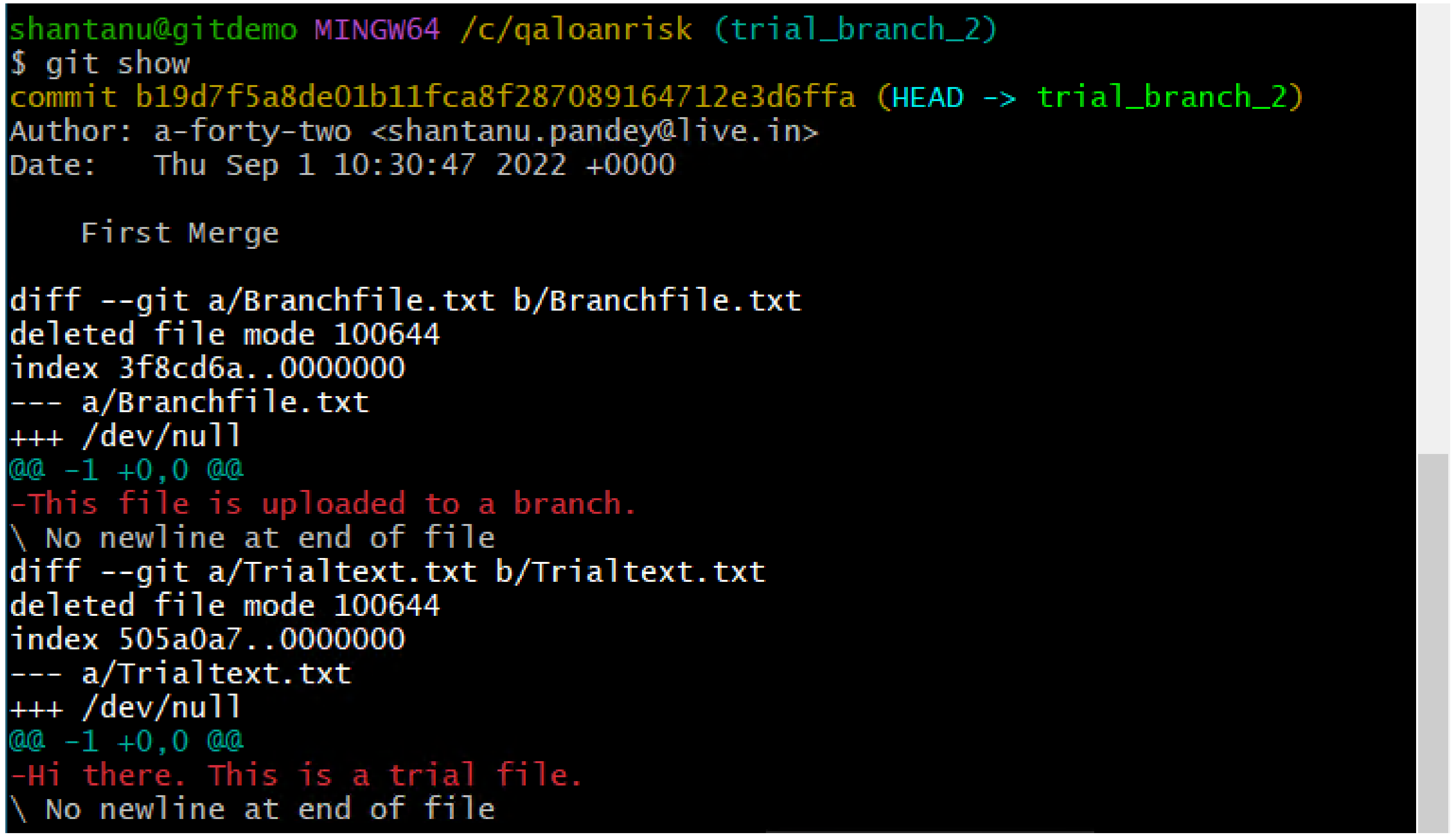
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git checkout master
Switched to branch 'master'

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git show
commit 6894a6b5a77b4c22f97ed23b05f4918a1b57d65f (HEAD -> master, trial_branch_
3, main)
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 08:33:35 2022 +0000

    This is a trial file.

diff --git a/Trialtext.txt b/Trialtext.txt
new file mode 100644
index 0000000..505a0a7
--- /dev/null
+++ b/Trialtext.txt
@@ -0,0 +1 @@
+Hi there. This is a trial file.
\ No newline at end of file
```

GIT SHOW



```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git checkout trial_branch
Switched to branch 'trial_branch'

shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ git show
commit 14a316ba38824a4e686728defa8f062d0bf55725 (HEAD -> trial_branch)
Merge: 86da750 b19d7f5
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:32:40 2022 +0000

    Merge branch 'trial_branch_2' into trial_branch
    This is a first merge


shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch)
$ |
```

GIT SHOW



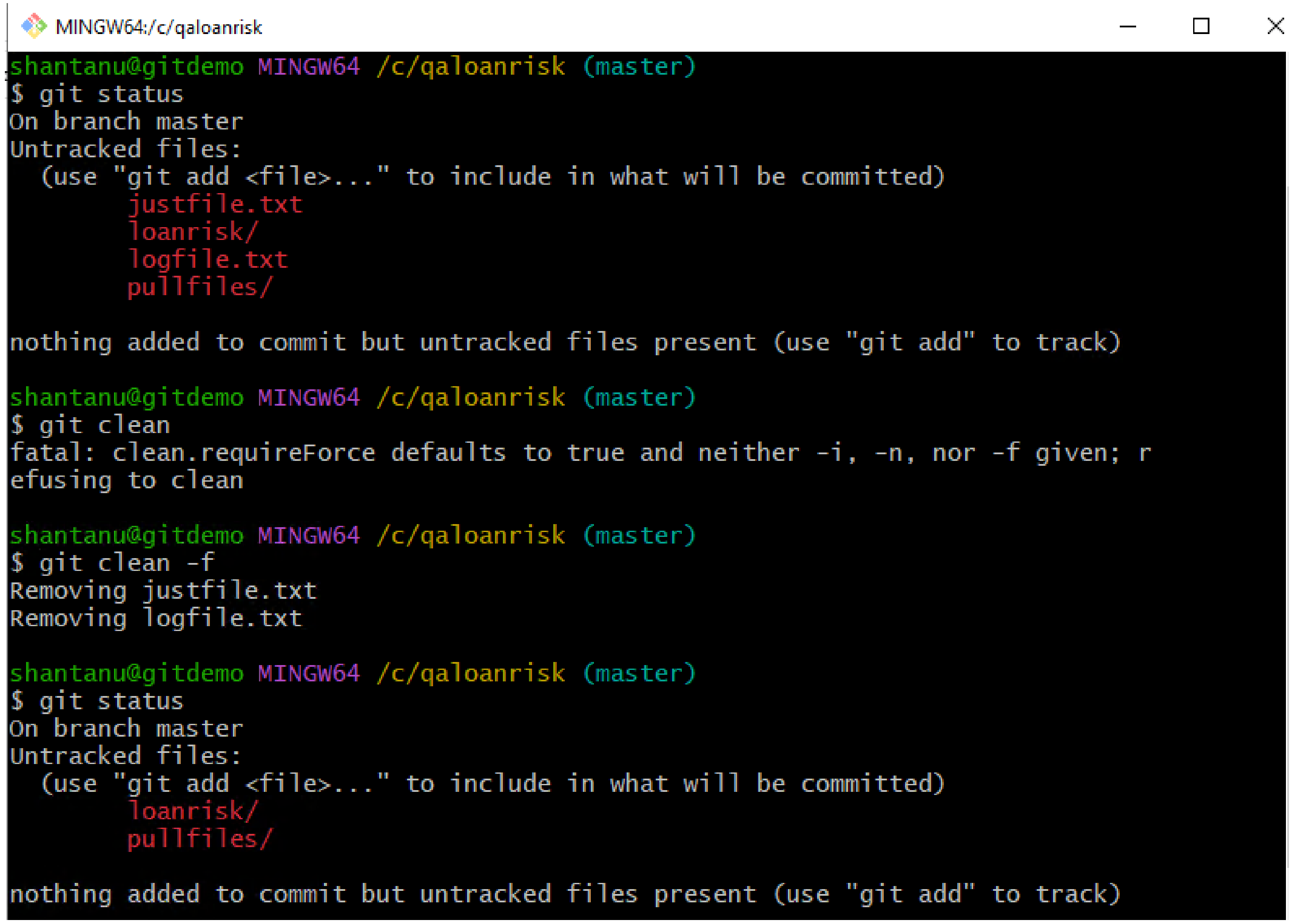
```
shantanu@gitdemo MINGW64 /c/qaloanrisk (trial_branch_2)
$ git show
commit b19d7f5a8de01b11fca8f287089164712e3d6ffa (HEAD -> trial_branch_2)
Author: a-forty-two <shantanu.pandey@live.in>
Date:   Thu Sep 1 10:30:47 2022 +0000

    First Merge

diff --git a/Branchfile.txt b/Branchfile.txt
deleted file mode 100644
index 3f8cd6a..0000000
--- a/Branchfile.txt
+++ /dev/null
@@ -1 +0,0 @@
-This file is uploaded to a branch.
\ No newline at end of file
diff --git a/Trialtext.txt b/Trialtext.txt
deleted file mode 100644
index 505a0a7..0000000
--- a/Trialtext.txt
+++ /dev/null
@@ -1 +0,0 @@
-Hi there. This is a trial file.
\ No newline at end of file
```

GIT CLEAN

git clean is used to remove untracked files, directories.

git clean –f or *git clean -d*

GIT CLEAN


MINGW64:/c/qaloanrisk

```
shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        justfile.txt
        loanrisk/
        logfile.txt
        pullfiles/

nothing added to commit but untracked files present (use "git add" to track)

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git clean
fatal: clean.requireForce defaults to true and neither -i, -n, nor -f given; r
efusing to clean

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git clean -f
Removing justfile.txt
Removing logfile.txt

shantanu@gitdemo MINGW64 /c/qaloanrisk (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        loanrisk/
        pullfiles/

nothing added to commit but untracked files present (use "git add" to track)
```

PRINCE2 foundation examination

Exam Content:

- Key concepts
- Principles
- Themes
- Processes

Exam format:

- 60 multiple choice questions
- 1 hour
- Pass mark – 55%, or 33 marks
- Closed book

Activity: (15 mins)
Create GitHub Account

- Work in groups of 2 or 3.
- Navigate to www.github.com on your web browser.
- Create an account.
- Create a new public repository.

2.1 A PROJECT IS...

“A temporary organisation that is created for the purpose of delivering one or more business products according to an agreed business case.”

5 Uses of Version Control

1. Change
2. Revert
3. Teamwork
4. Branch and Merge
5. Test and Deploy

POWERING
POTENTIAL

THANK YOU

Hope you enjoyed this learning journey.

POWERING
POTENTIAL

DevOps and Version Control

2017 Version

Housekeeping

Couse Outline

- History of Devops
- Why Devops needs automation
- CI/CD
- Benefits of CI/CD
- Ci/CD & Iterative development
- How agile fits into CI/CD cycle
- A typical build pipeline
- Infrastructure as code
- Configuration as code

COURSE OUTCOMES

- Demonstrate an understanding of the various components of the PRINCE2 method.
- Describe each step of the PRINCE2 journey, listing the activities required to guide a project throughout its lifecycle.
- Outline the relevant aspects of PRINCE2 principles, themes and processes in context.

HISTORY OF DEVOPS

- DevOps has grown to exert a revolutionary influence in the area of software development and deployment through constant innovation and development.

- During a conversation between Andrew Clay and Patrick Debois in 2008, the idea of DevOps was born.

- They sought to develop a better solution since they were worried about Agile's shortcomings.

HISTORY OF DEVOPS

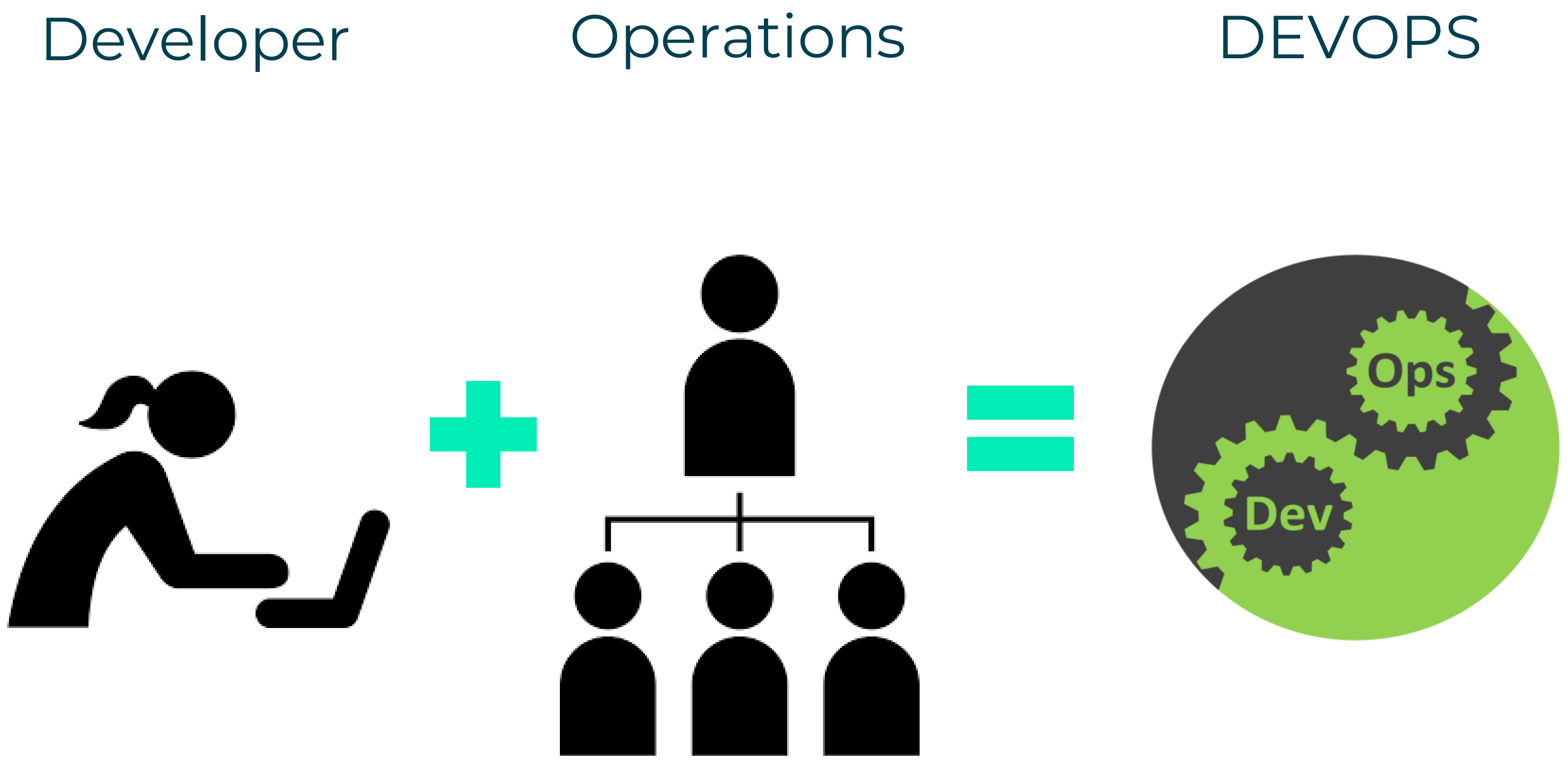
Developer

Operations

HISTORY OF DEVOPS

- Developers and operations managers came together to share their thoughts and worries about the business and the best ways to complete work, and this is how DevOps was created.
- The strength of DevOps is in the culture that fosters it and helps break down the habit of silos.

HISTORY OF DEVOPS



AUTOMATION IN DEVOPS

- Automation is the utmost requirement for DevOps practise, and the guiding philosophy of DevOps is to "automate everything."
- DevOps automation is the use of technology to augment processes that enable feedback loops between operations and development to hasten the deployment of iterative changes to applications in production.
- The entire DevOps lifecycle, including live application performance monitoring, continuous integration, continuous testing, and continuous deployment, is automated.

WHY DEVOPS NEEDS AUTOMATION

- DevOps automation promotes efficiency, consistency, reliability, and more delivery.
- With automation, DevOps can ensure consistency across repetitive processes while minimising the risk of human error.
- Overall, DevOps enables all development and operational activities to be automated wherever possible or when tasks are repeatable, and when accuracy is required.

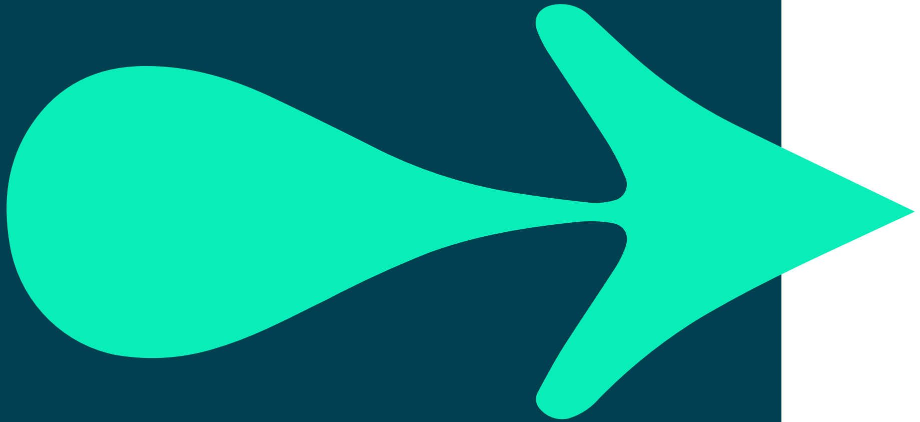
CI/CD

CONTINUOUS INTEGRATION

- Continuous integration and continuous delivery is a best practice in Devops and in Agile methodology.

- Continuous integration makes it easier to find and repair faults and security problems, and it does this considerably faster in the software development lifecycle.

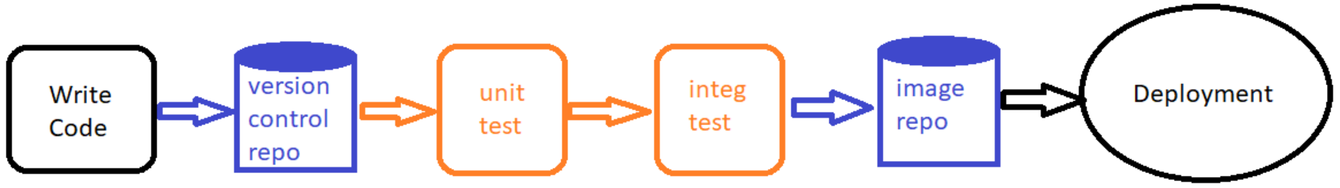
- Even when several developers are working on the same application, code conflict can be minimised by often merging changes and starting automatic testing and validation procedures.

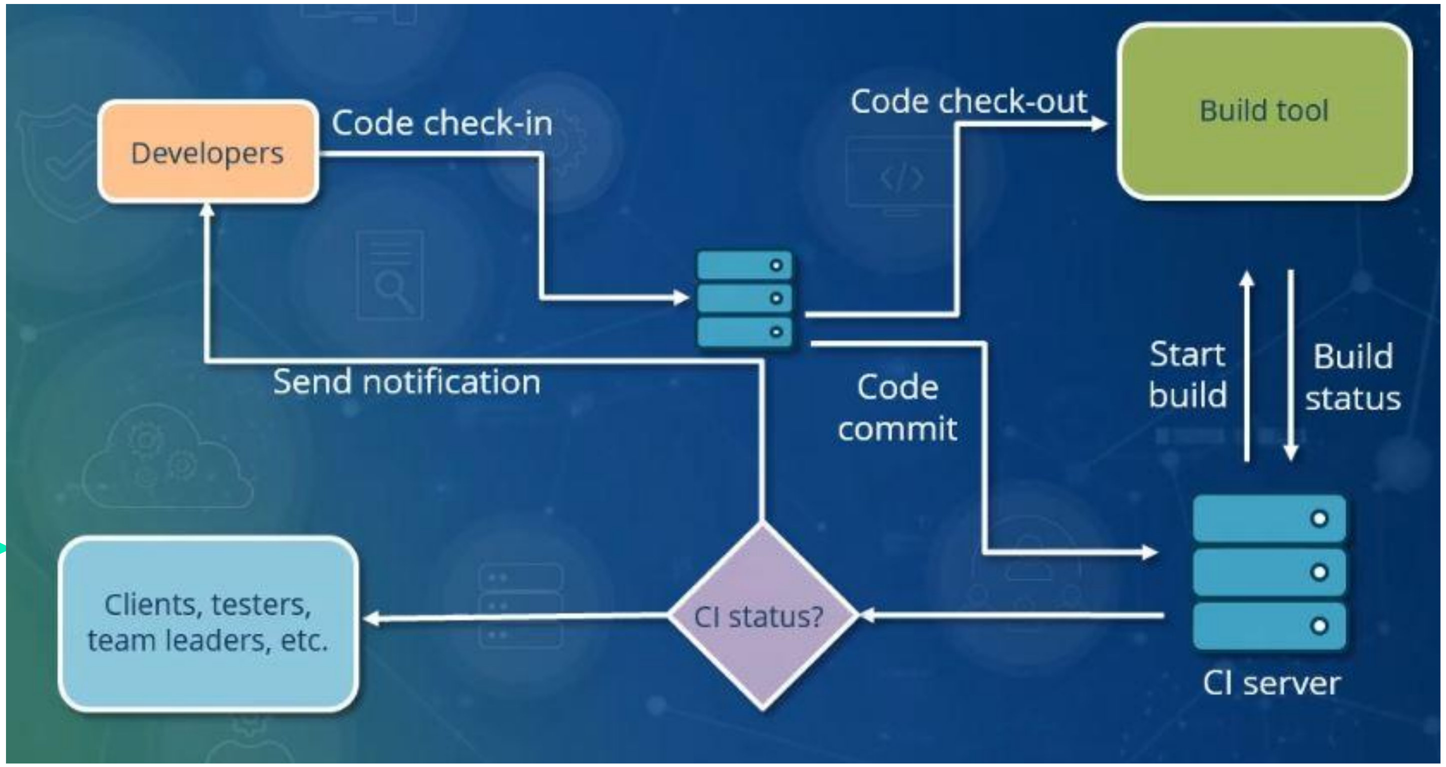


CI/CD

CONTINUOUS INTEGRATION

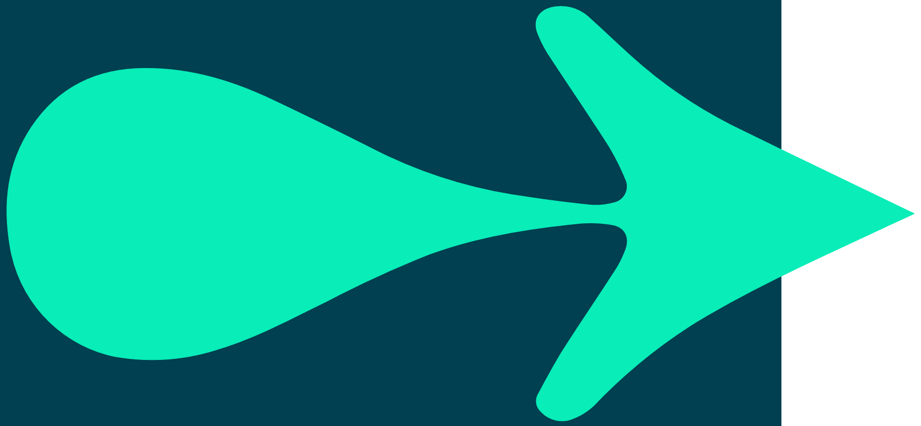
Continuous Integration





CI/CD

CONTINUOUS DELIVERY

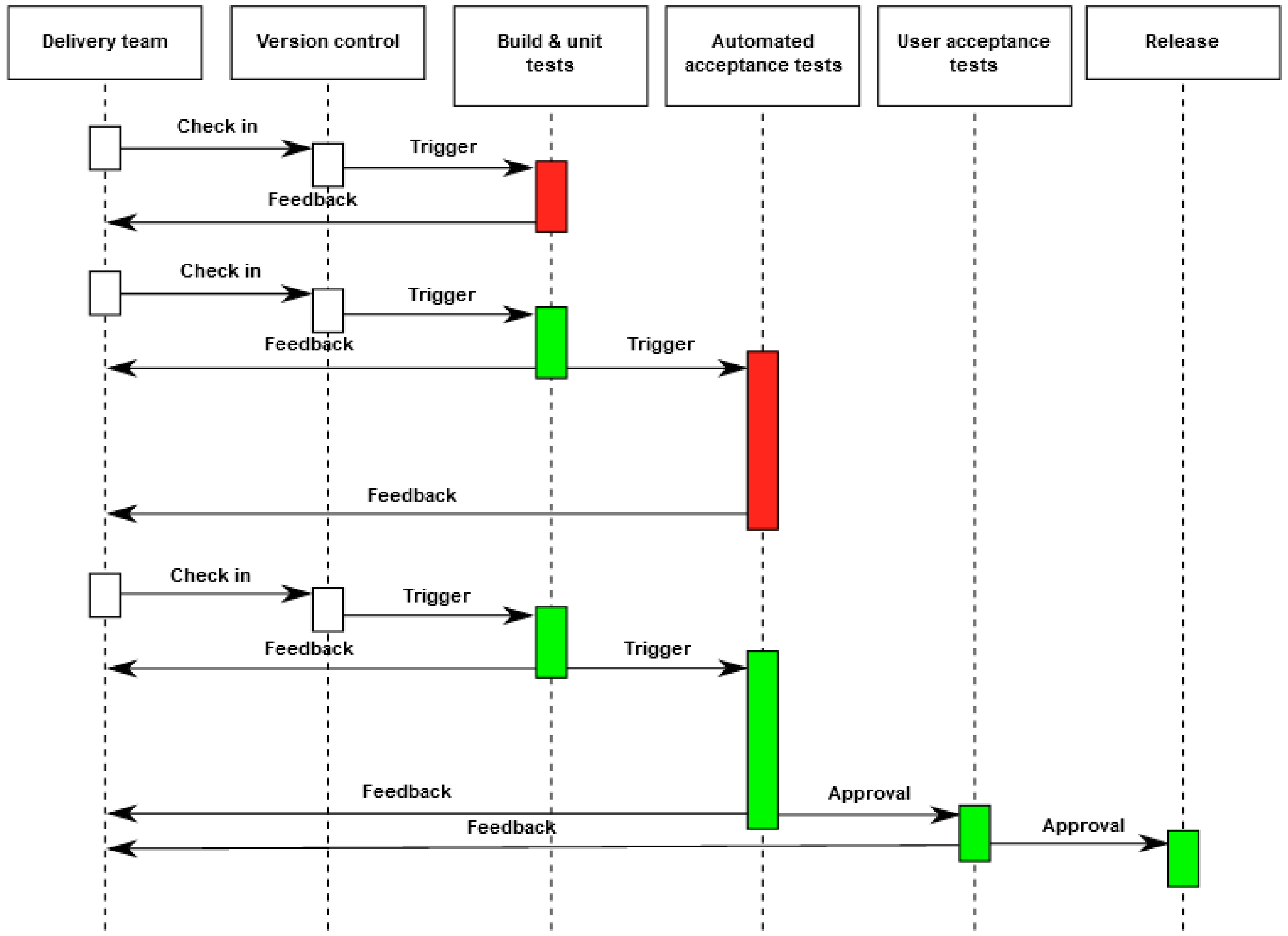


- Continuous delivery is an expansion of continuous integration.
- It automatically distributes all code changes to a testing and / or production environment following the build step.
- With continuous delivery, you can choose to distribute daily, weekly, biweekly, or whenever the requirement is necessary for your business.
- You should deploy to production as soon as you can if you want to reap the benefits of continuous delivery.

CI/CD

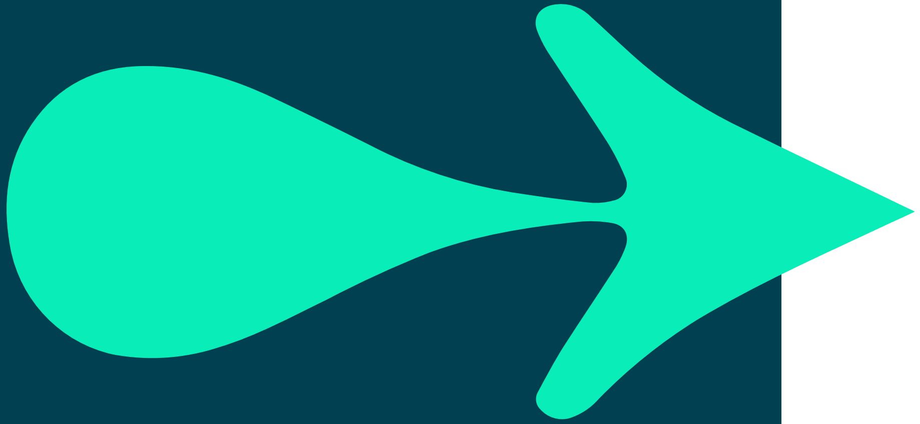
CONTINUOUS DELIVERY

Continuous Delivery



CI/CD

CONTINUOUS DEPLOYMENT

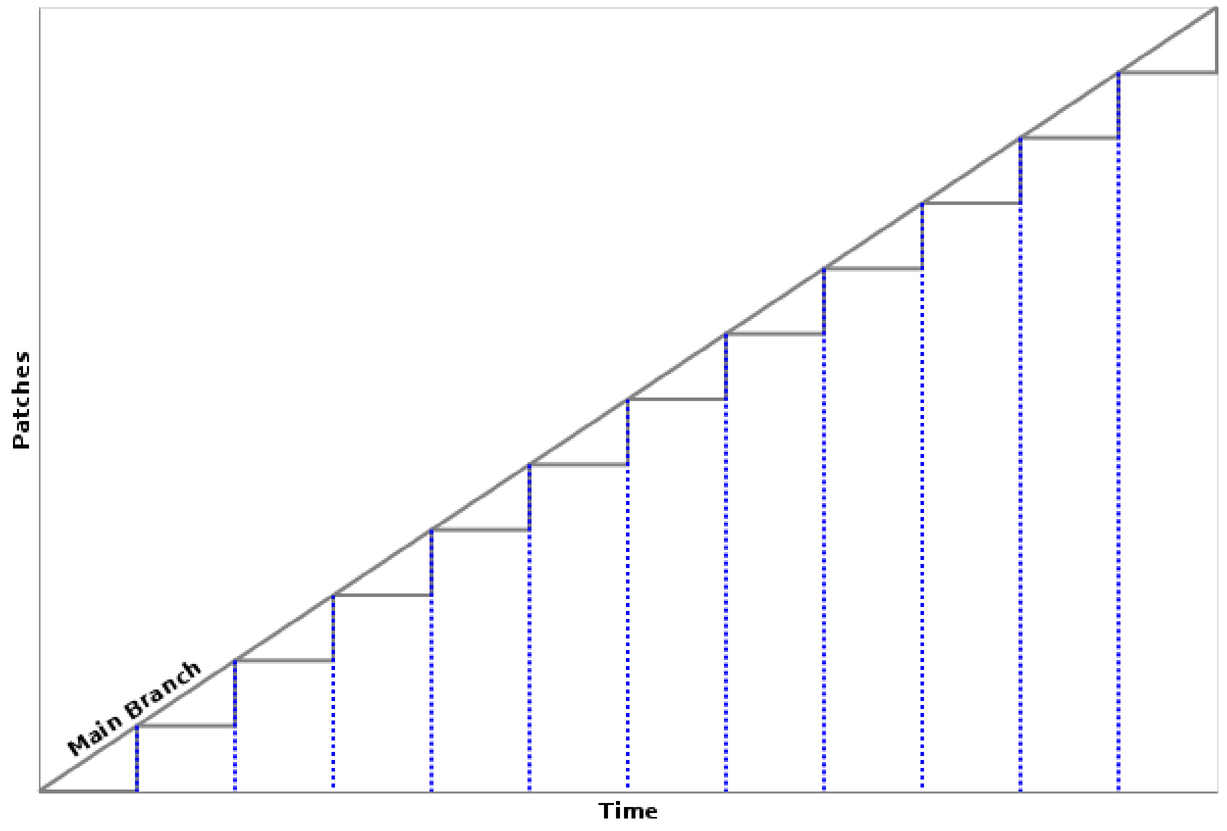


- Continuous deployment is the final phase of an effective CI/CD process.
- Organisations can deploy their applications automatically with the help of continuous deployment, terminating the requirement for human interference.
- Continuous deployment largely relies on well-designed test automation since there is no manual gate at the pipeline level prior to production.
- Continuous deployment is a brilliant way to shorten the feedback loop with your customers and relieve team stress.

CI/CD

CONTINUOUS DEPLOYMENT

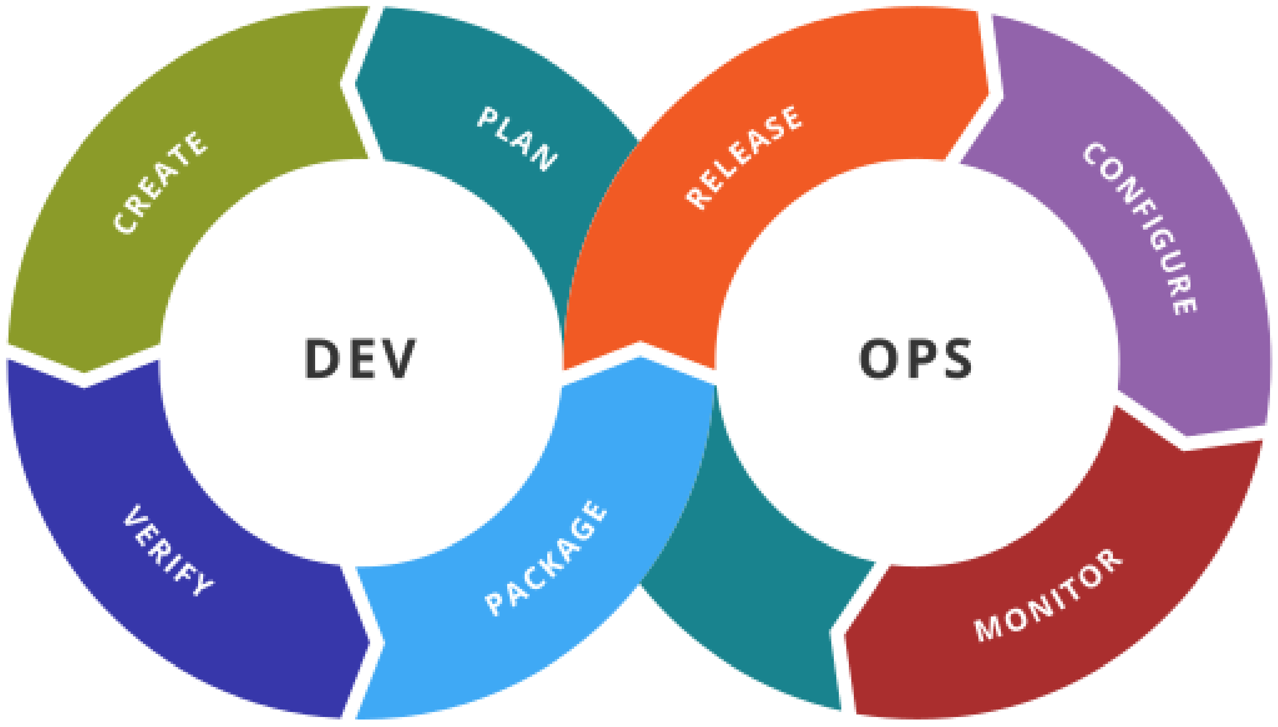
Continuous Deployment



CI/CD

CONTINUOUS DEPLOYMENT

CI + CD + Deployment = DEVOPS



BENEFITS OF CI/CD

FASTER TIME TO MARKET

- Organisations are aiming for multiple daily feature releases.
- Failures are discovered more quickly, which enables quicker repairs and ultimately higher release rates.
- A quicker turnaround lowers your development costs and frees up your workforce for additional tasks. Customers benefit from quicker outcomes and a competitive advantage.

IMPROVED CODE QUALITY

- While extensively testing your code's behaviour can take a lot of time, it is a necessary stage in the product release process.

- Regular and thorough testing can help you find flaws in your code earlier and make it simpler to resolve them because less functionality will have been added on top of them.

IMPROVED CODE QUALITY

- This leads to higher-quality code over time.
- Testing is performed consistently when it is automated, increasing the reliability of the outcomes when compared to manual counterparts.

REDUCED RISK

- Defects are costly and time-consuming to find and fix, much later in the development process.

- This is particularly true when features that have already been released to production have problems.

REDUCED RISK

- A CI/CD pipeline enables you to test and release code more frequently, enabling testers to find problems as soon as they arise and to solve them right away.

- Early and frequent user testing of your innovations allows you to confirm your strategy before spending months or years developing a product that doesn't genuinely address a problem for your users.

COLLABORATION & COMMUNICATION

- The beginning of a positive feedback loop is tearing down the barriers between development and operations.
- Building a collaborative culture is as important to DevOps as implementing new procedures and tools.

COLLABORATION & COMMUNICATION

- You must start removing bottlenecks between teams and promoting increased collaboration if you want to utilise CI/CD.
- Sharing information about upcoming releases, usage statistics, and experiment results encourages greater communication, which in turn encourages innovation.

EFFICIENT INFRASTRUCTURE

- Automating the construction of environments is a component of the infrastructure-as-code strategy.
- To prevent the risk of unintentional modifications and inconsistencies, individual servers' configurations are programmed and kept in version control rather than being managed manually.

EFFICIENT INFRASTRUCTURE

- This allows for the speedy deployment of new environments.
- As a result, the continuous delivery stage is expedited and strengthened.

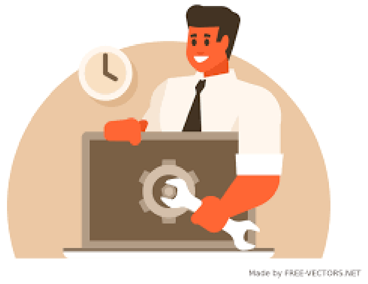
ENHANCED CREATIVITY & FREE TIME

- An automated system also frees up people to be creative by employing computers to carry out tedious chores.
- Developers can concentrate on resolving issues and experimenting with solutions rather than adhering to manual test scripts, resetting environments, or delivering upgrades.
- The team is more productive and has time for more fulfilling initiatives thanks to the deployment process' increased automation.

REDUCED COSTS

- Automation in the CI/CD pipeline lowers the possibility of errors in the numerous recurring CI and CD phases.
- Additionally, by catching the problem as soon as possible, less code modifications will need to be made in the future to repair it, which frees up developer time that may be used for product development.
- Another thing to consider is that improving code quality through automation also improves return on investment.

FASTER BUG FIXES



- Automated testing has helped to improve code quality, but defects will occasionally make it to production.
- Each release to production will have a relatively modest amount of code changes if you're routinely committing changes and shipping them, which will make it much simpler to pinpoint the source of a problem.
- Running automated tests no longer represents a substantial overhead with a CI/CD pipeline, which reduces the temptation to overlook quality issues.

MEASURABLE PROGRESS

- Many of the CI/CD pipeline support tools instrument the process, giving you access to a wide range of metrics such as build durations, test coverage, defect rates, and test fix times.
- Metrics can help you improve your code test coverage, lower your defect rate, and release more frequently.
- Another benefit of the process is the ability to track how your organisation's objectives are being supported by your CI/CD pipeline.

EASY MAINTENANCE & UPDATES

- Making a great product requires regular maintenance and upgrades.
- It's crucial to remember that in a CI/CD process, maintenance should be carried out during downtime periods.

EASY MAINTENANCE & UPDATES

- Integrate the release and change dates into the pipeline to ensure smooth operation.
- Making use of microservices in your code design will enable you to limit the impact of maintenance to only one region of the system at a time.

HOW CI/CD ALLOWS ITERATIVE DEVELOPMENT

- Utilising continuous delivery, you may successfully and frequently launch new product value in digestible chunks to minimise risk.
- You can deliver continuously when you create iteratively.

Listed in the upcoming slides are some of the important reasons why iterative software development is important.

BENEFIT

Faster delivery of value to the user.

BENEFIT

Reduces the risk of sending non-value-adding items.

BENEFIT

Easier to identify and comprehend the effects of the modifications.

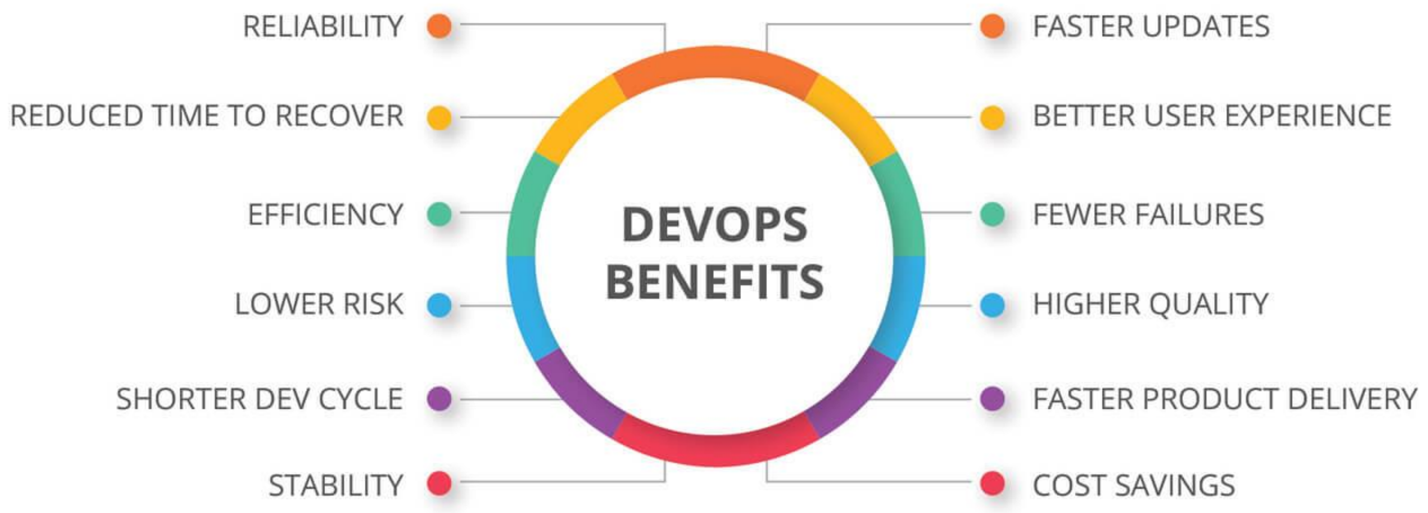
BENEFIT

Faster shipping so the team can start learning faster.

BENEFIT

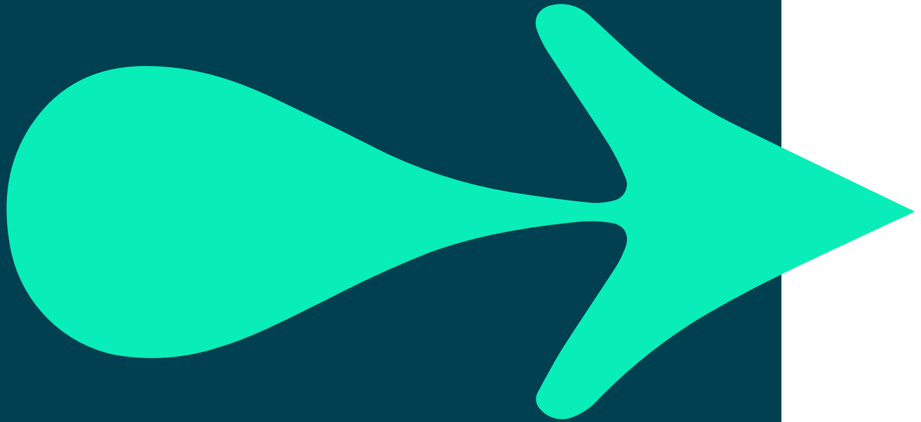
Give teams more time to consider future iterations or opt to drop the trial earlier. This would in turn save time and resources.

BENEFIT

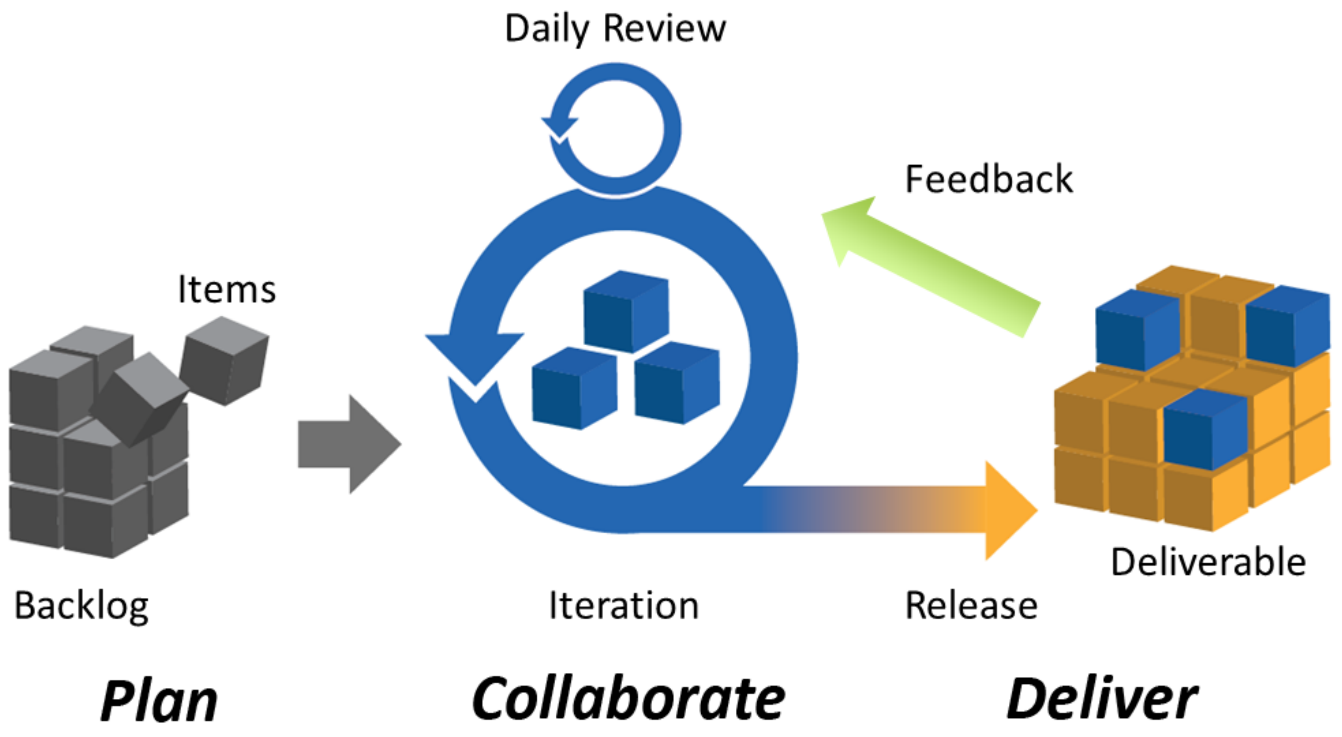

RELIABILITY
REDUCED TIME TO RECOVER
EFFICIENCY
LOWER RISK
SHORTER DEV CYCLE
STABILITY
DEVOPS BENEFITS
FASTER UPDATES
BETTER USER EXPERIENCE
FEWER FAILURES
HIGHER QUALITY
FASTER PRODUCT DELIVERY
COST SAVINGS

WHAT IS AGILE?

- Agile development is essential for producing high-quality, timely releases of applications.
- Many firms employ continuous integration (CI), continuous delivery (CD), and continuous testing (testing) in addition to Agile to implement this development process.



AGILE



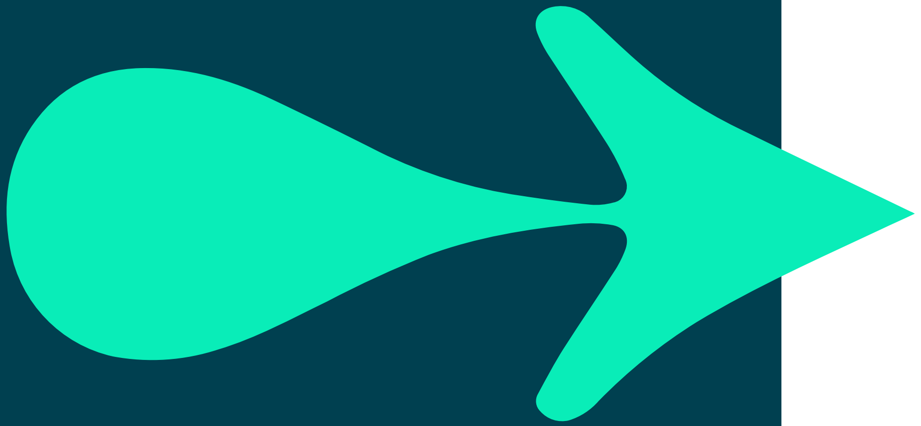
Agile Project Management: Iteration

CORE CONCEPTS OF AGILE

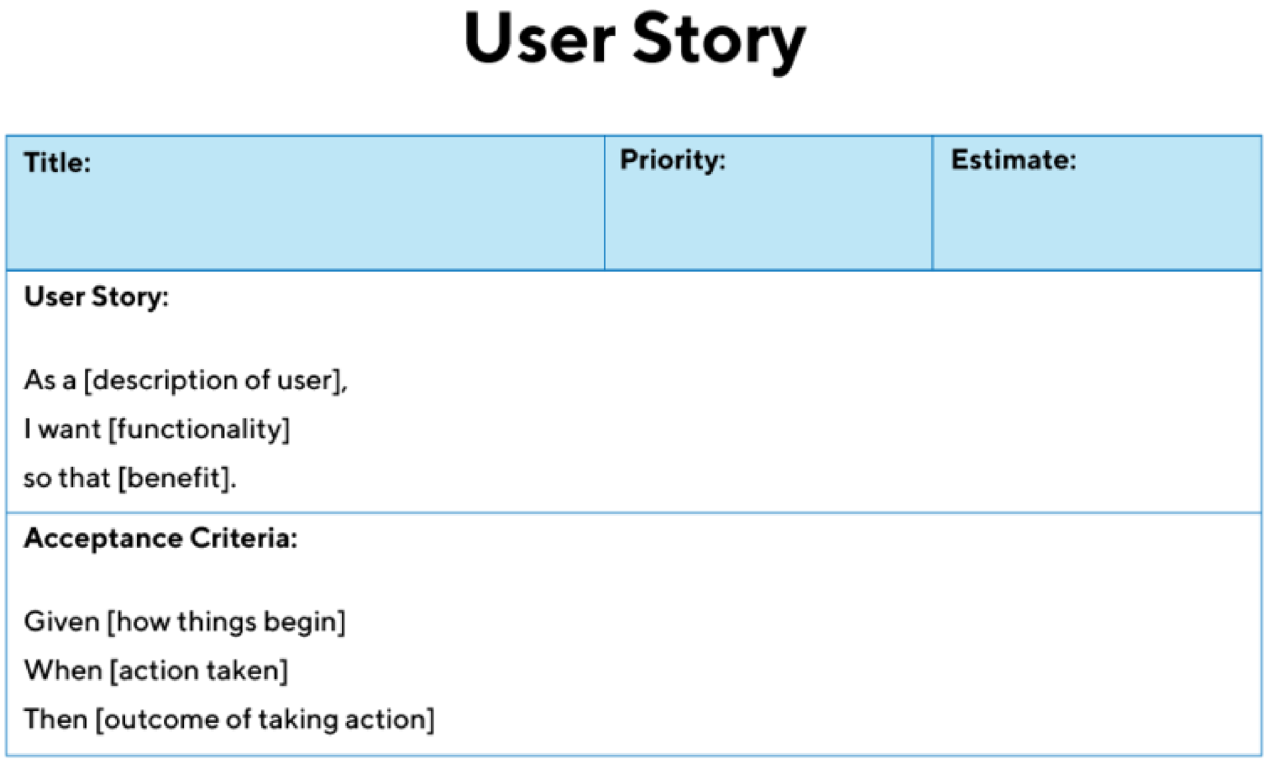
By emphasising incremental change, Agile project management completes challenging tasks. The six key components of this iterative technique are used to monitor progress and develop the final result.

USER STORIES

Outlines the product's objectives from the viewpoint of the user.



USER STORIES

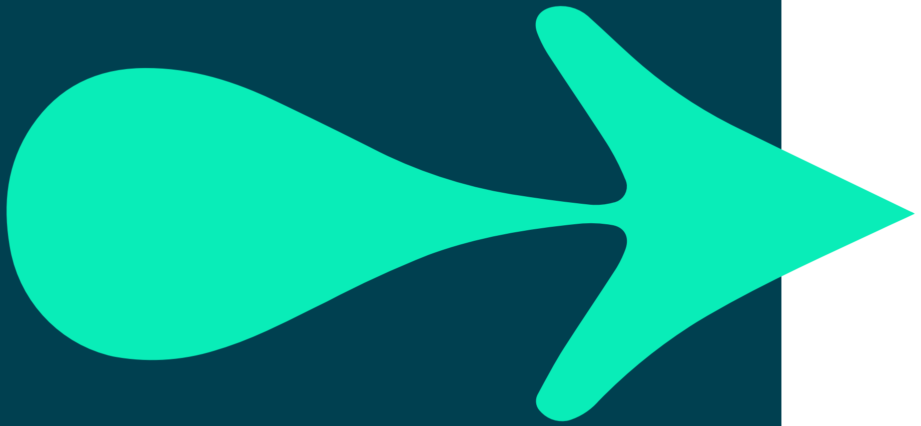


User Story

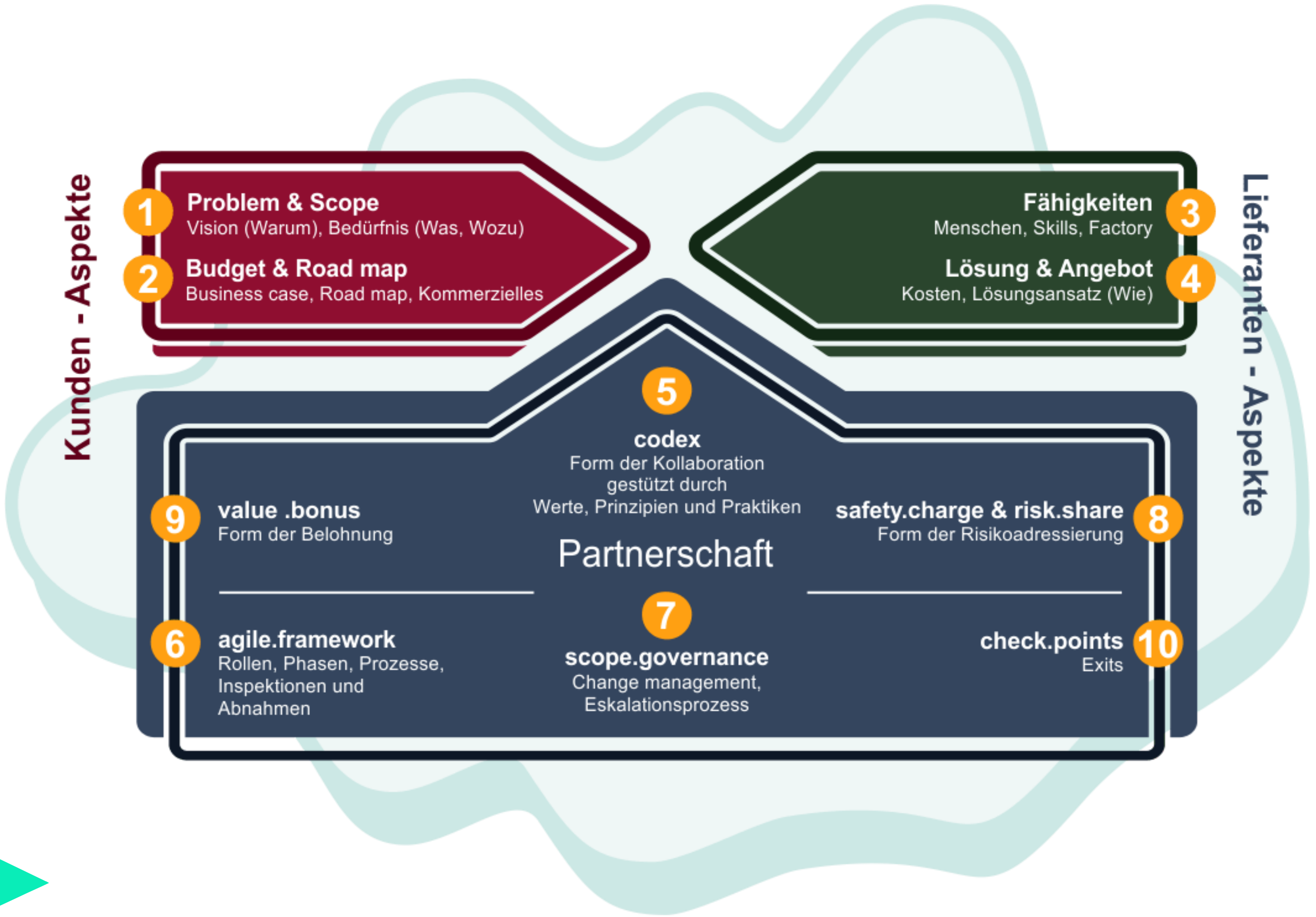
Title:	**Priority:**	**Estimate:**
User Story: As a [description of user], I want [functionality] so that [benefit].		
Acceptance Criteria: Given [how things begin] When [action taken] Then [outcome of taking action]		

ROADMAP

A broad perspective of the demands are necessary to realise the product goal.

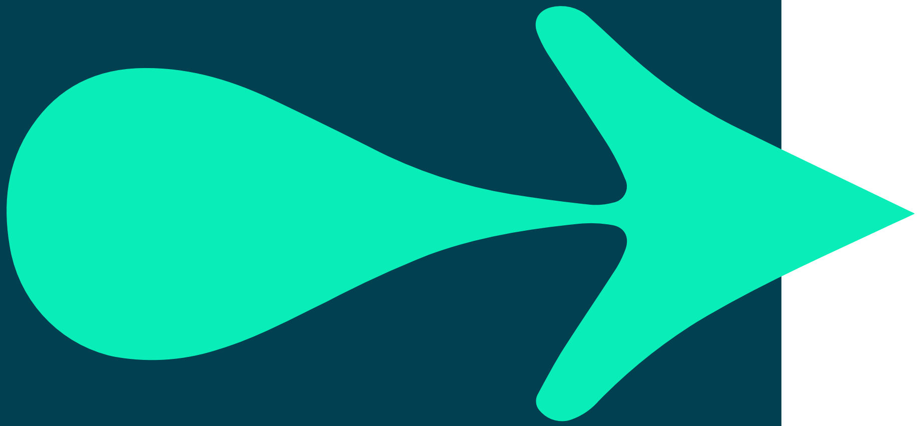


ROADMAP

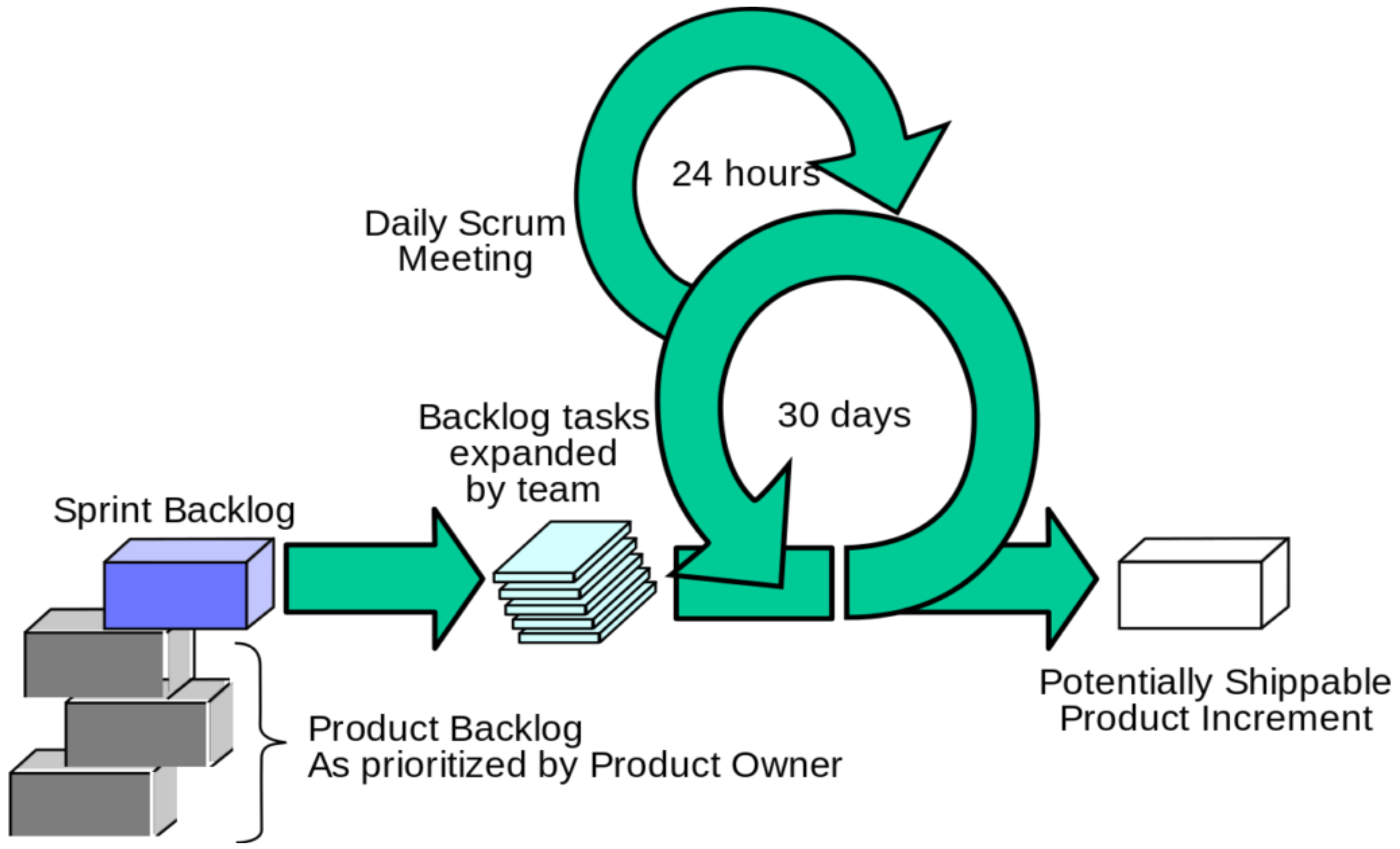

Kunden - Aspekte
1 Problem & Scope
Vision (Warum), Bedürfnis (Was, Wozu)
2 Budget & Road map
Business case, Road map, Kommerzielles
Fähigkeiten 3
Menschen, Skills, Factory
Lösung & Angebot 4
Kosten, Lösungsansatz (Wie)
Lieferanten - Aspekte
5 codex
Form der Kollaboration gestützt durch Werte, Prinzipien und Praktiken
9 value .bonus
Form der Belohnung
safety.charge & risk.share 8
Form der Risikoadressierung
Partnerschaft
6 agile.framework
Rollen, Phasen, Prozesse, Inspektionen und Abnahmen
7 scope.governance
Change management, Eskalationsprozess
check.points 10
Exits

BACKLOG

Project requirements listed in order of priority.

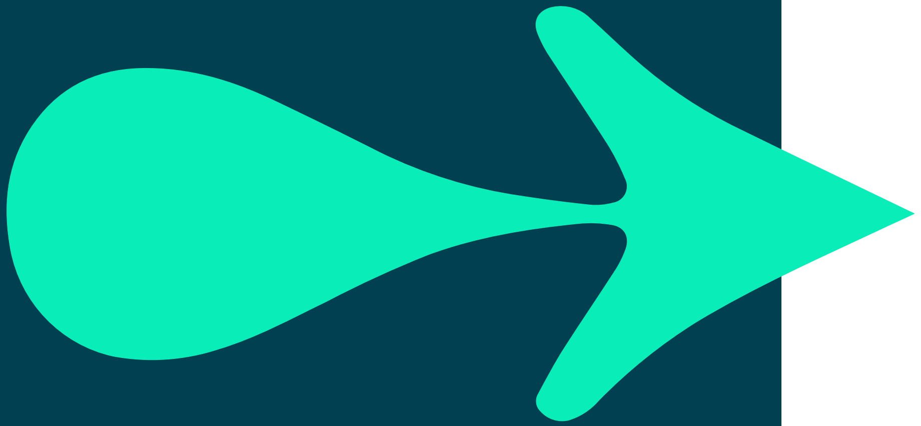


BACKLOG


24 hours
Daily Scrum Meeting
30 days
Backlog tasks expanded by team
Sprint Backlog
Product Backlog As prioritized by Product Owner
Potentially Shippable Product Increment

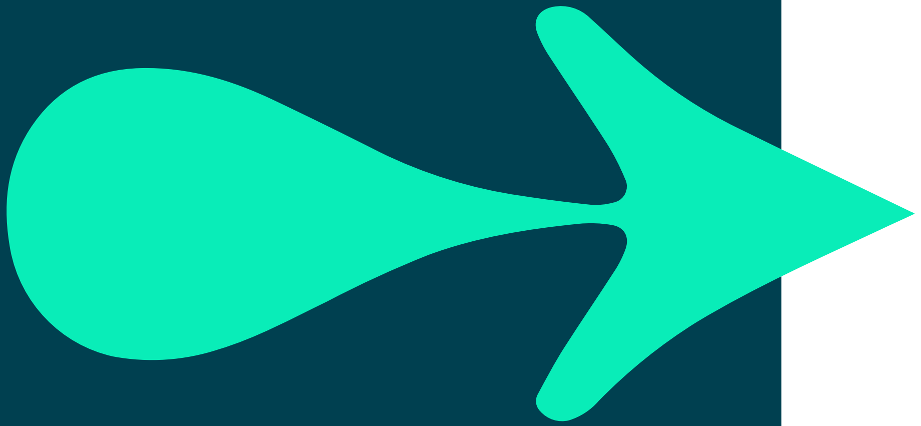
RELEASE PLAN

A timeline for the delivery of a functional product.

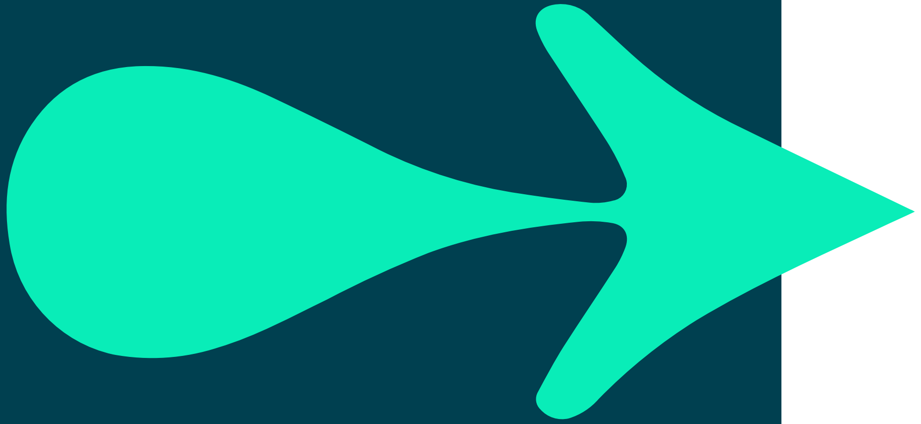


SPRINT

The tasks, objectives, and user stories associated to a given sprint.



RISK MANAGEMENT

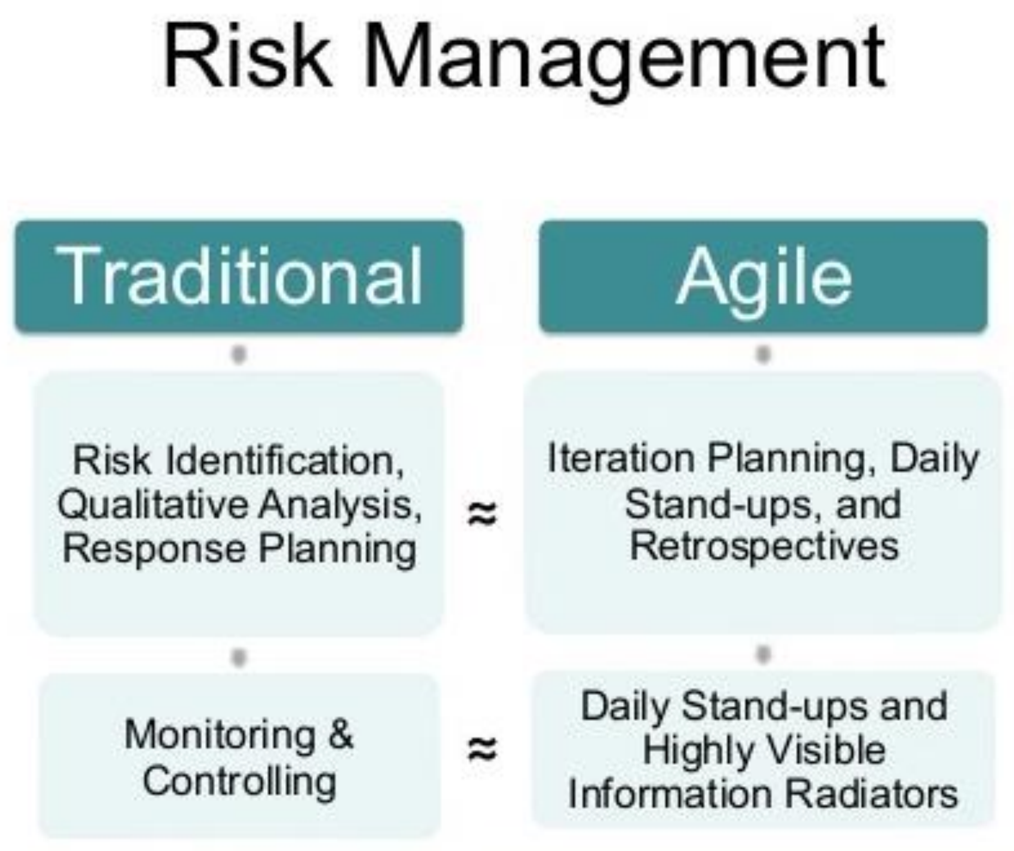


Agile exposes and provides the opportunity to recognise and mitigate risk early. Risk mitigation is achieved through:

- **cross-functional teams**
- **sustainable and predictable delivery pace**
- **continuous feedback**
- **good engineering practices**

Transparency at all levels of an enterprise is also key.

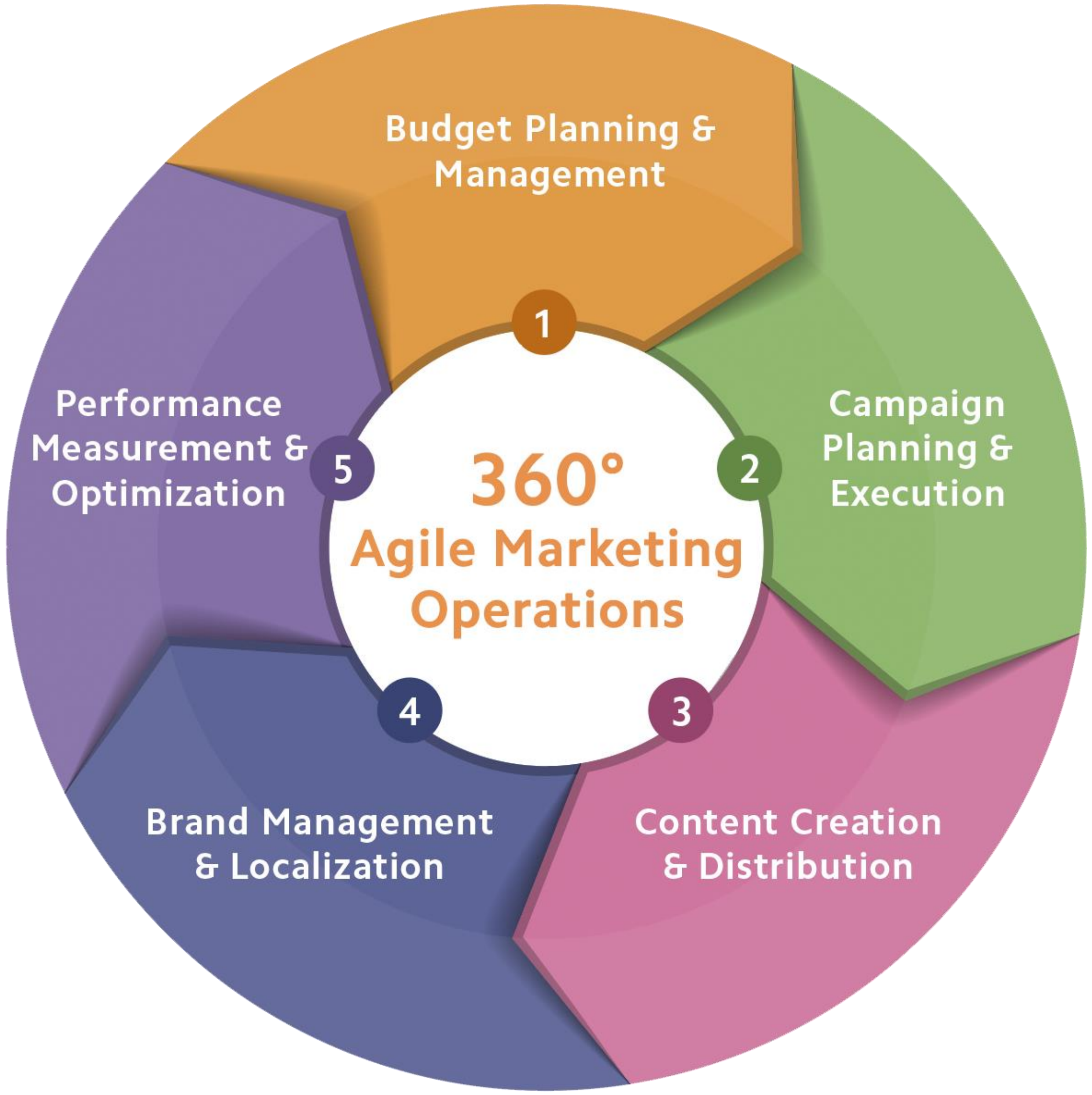
RISK MANAGEMENT


Risk Management
Traditional
Agile
Risk Identification, Qualitative Analysis, Response Planning
≈
Iteration Planning, Daily Stand-ups, and Retrospectives
Monitoring & Controlling
≈
Daily Stand-ups and Highly Visible Information Radiators

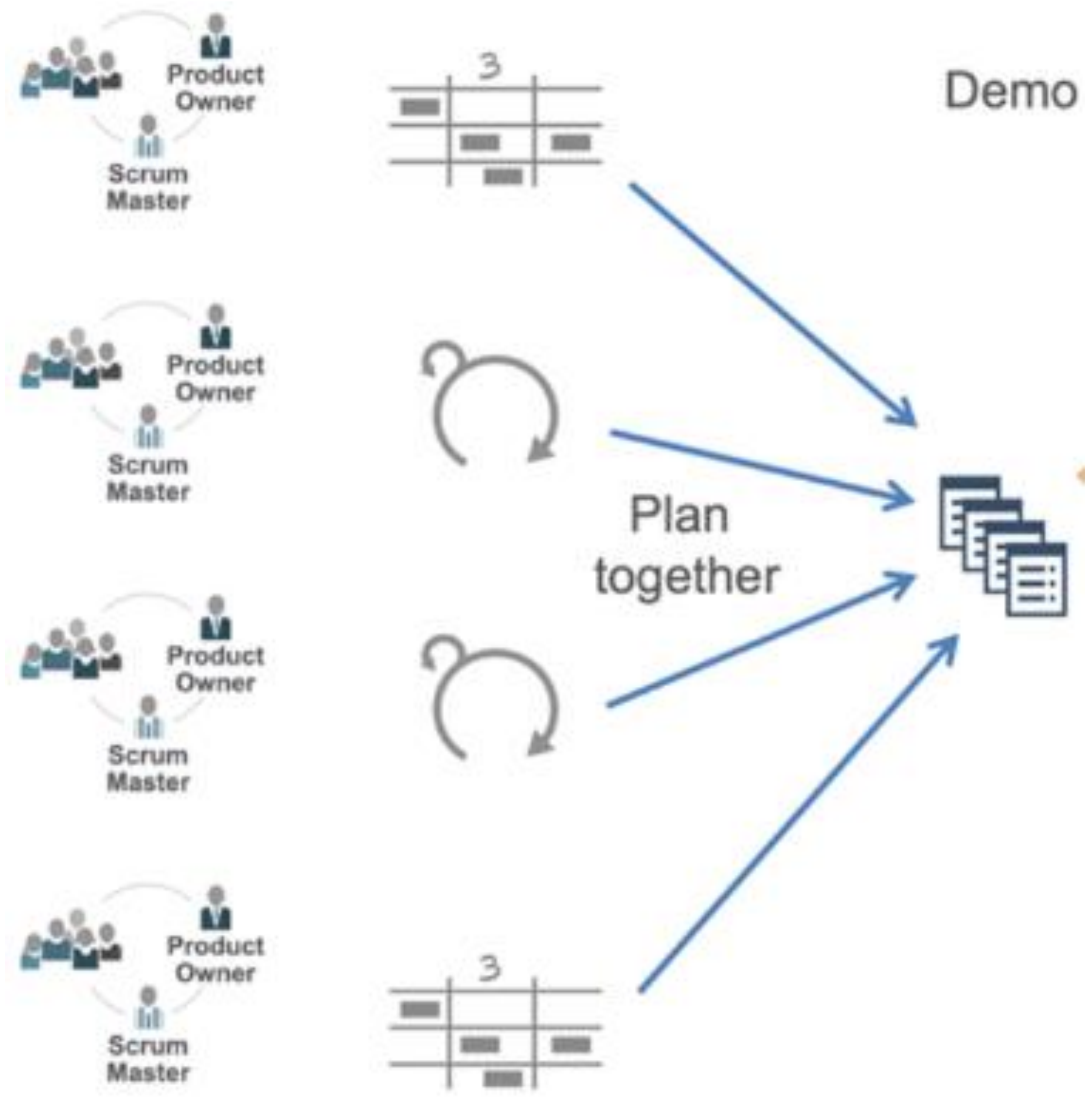
INCREMENT

The finished product that is demonstrated to the stakeholders after a Sprint.

AGILE MARKETING EXAMPLE


Budget Planning & Management
1
Campaign Planning & Execution
2
Content Creation & Distribution
3
Brand Management & Localization
4
Performance Measurement & Optimization
5
360° Agile Marketing Operations

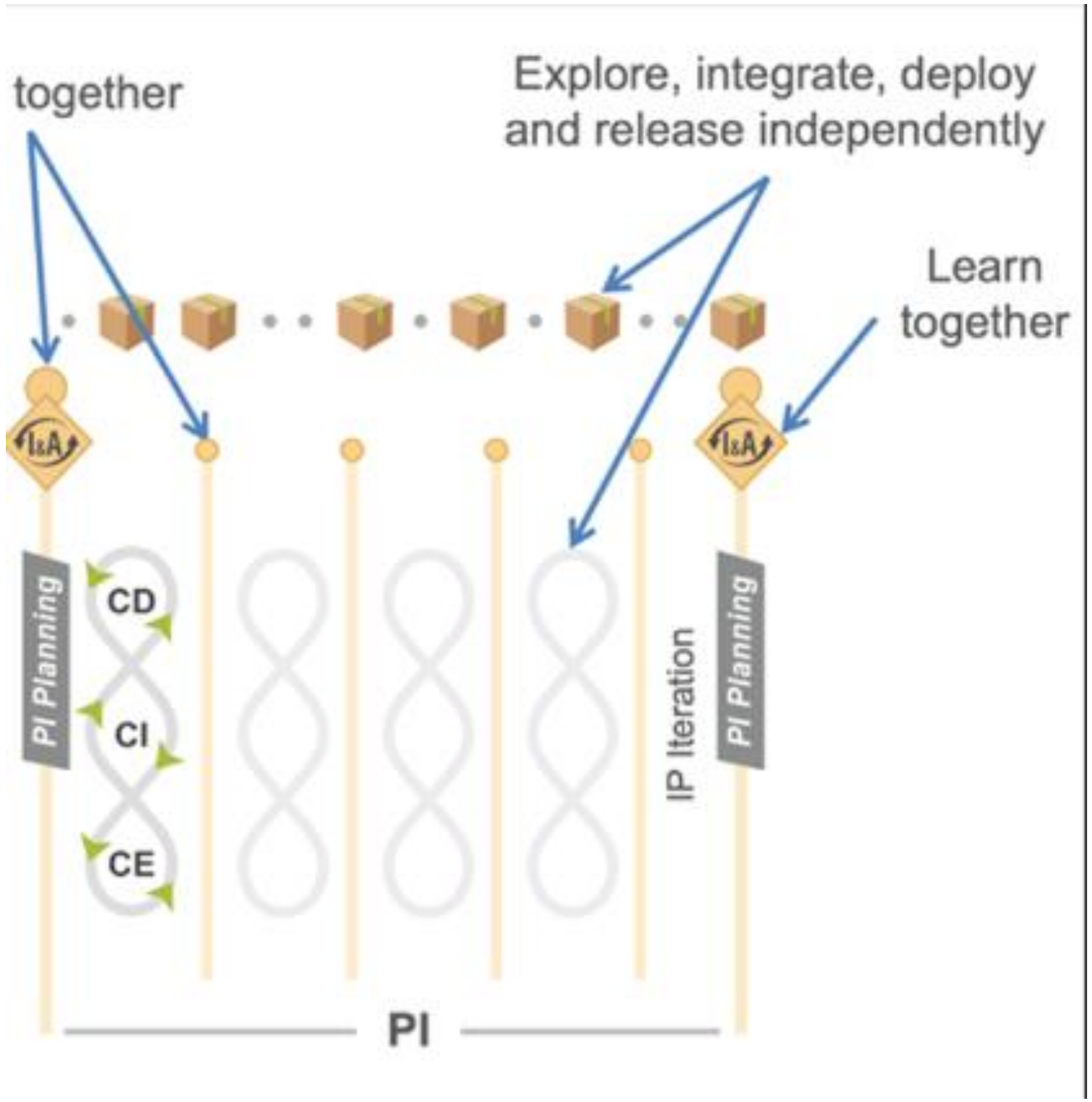
AGILE TEAMS EXAMPLE



Next

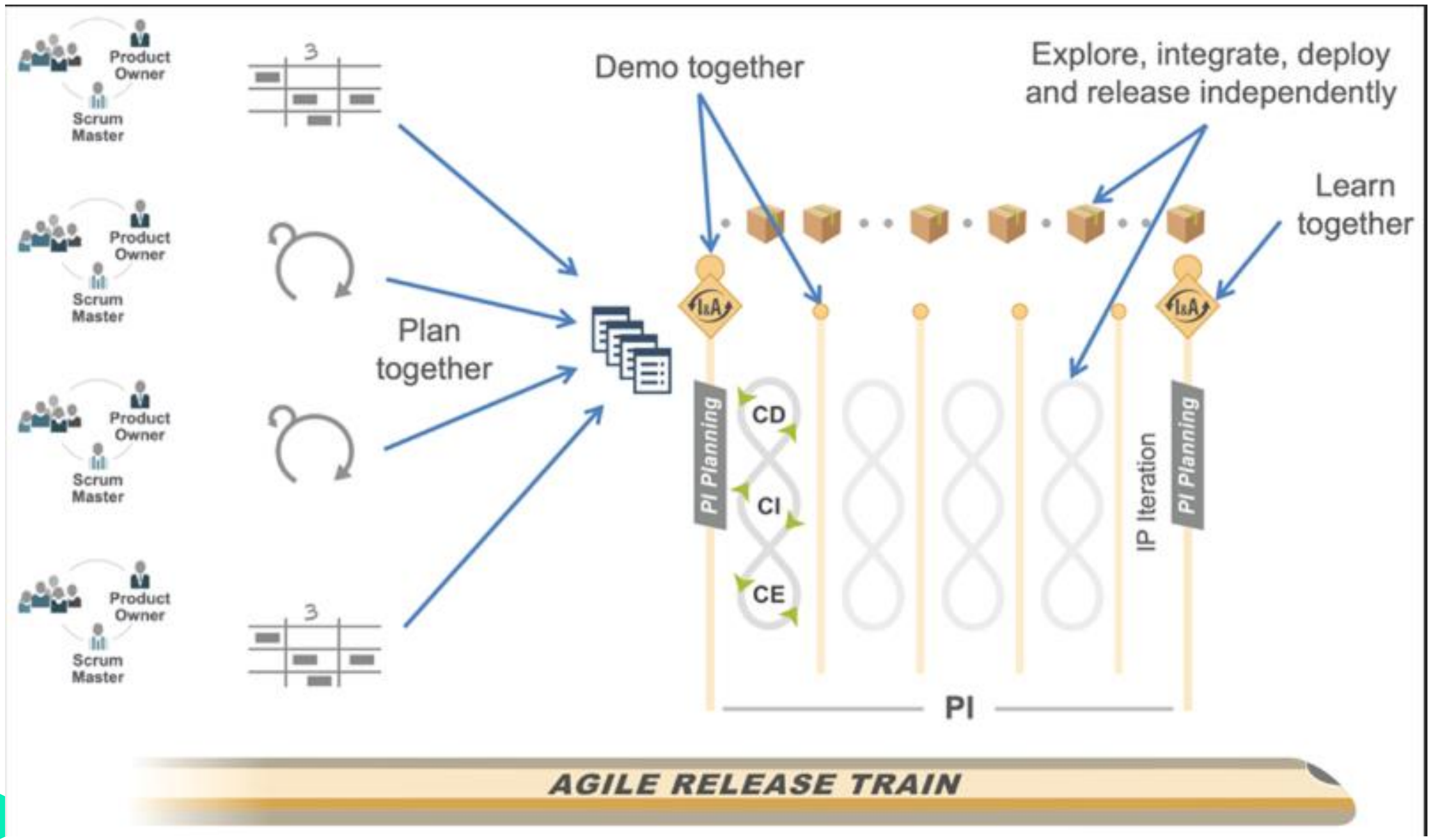
AGILE TEAMS EXAMPLE

Previous



AGILE TEAMS EXAMPLE

From slide no. 55+56



HOW AGILE FITS INTO CI/CD CYCLE

- The foundation of agile is the belief that developers may produce minor, incremental updates to a good or service.

- This can only be done in practice if an organisation makes a commitment to CI/CD automation.

HOW AGILE FITS INTO CI/CD CYCLE

- Teams may operate autonomously and swiftly with Agile frameworks.
- At the end of each Sprint, Agile development aims to have a usable product.

HOW AGILE FITS INTO CI/CD CYCLE

- With CI, product code changes are continuously compared with other changes.

- Teams can produce a working product since flaws and potential problems are found before they affect the end user thanks to continuous integration and testing.

HOW AGILE FITS INTO CI/CD CYCLE

- Time to market is critical!

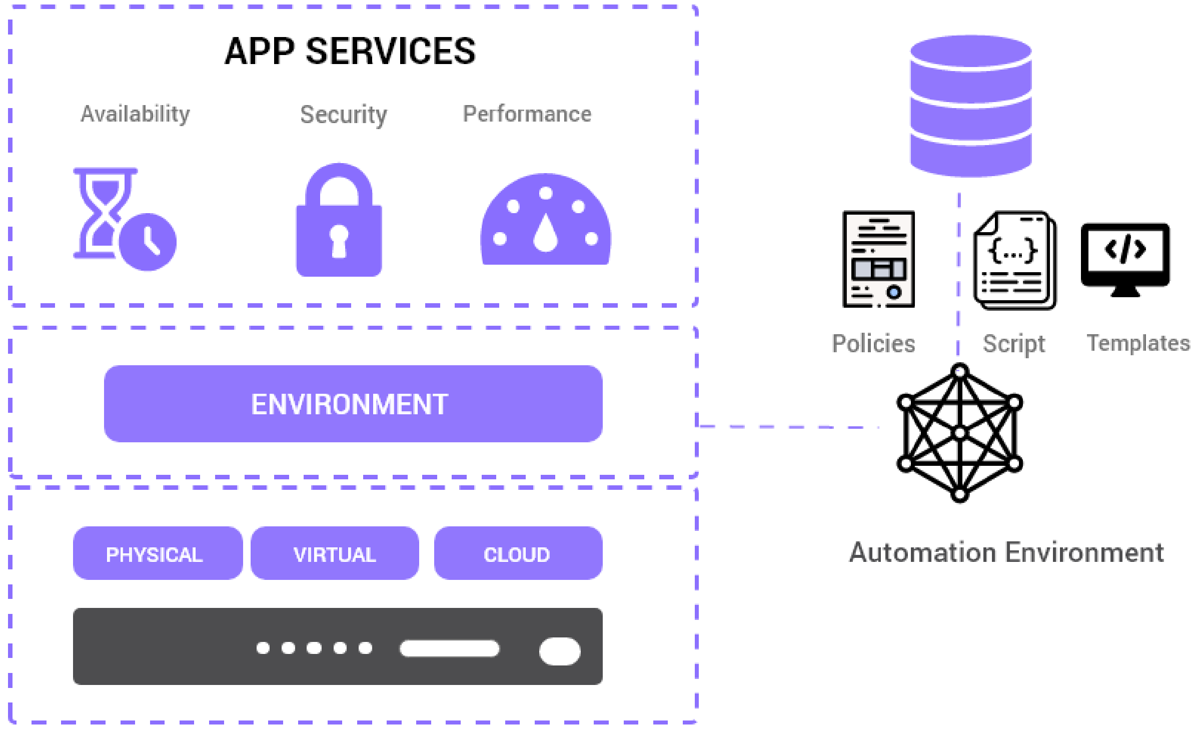
- CI/CD in Agile is the solution to this. They support the desired speed and quality requirements.

INFRASTRUCTURE AS CODE

- The management of networks, virtual machines, load balancers, and connection topologies in a descriptive model is known as infrastructure as code.
- It employs the same versioning system for source code as a DevOps team does.
- An IaC model creates the same environment each time it is used, similar to how a key can only open a single door.
- It is used in conjunction with continuous delivery and is crucial to the DevOps process.

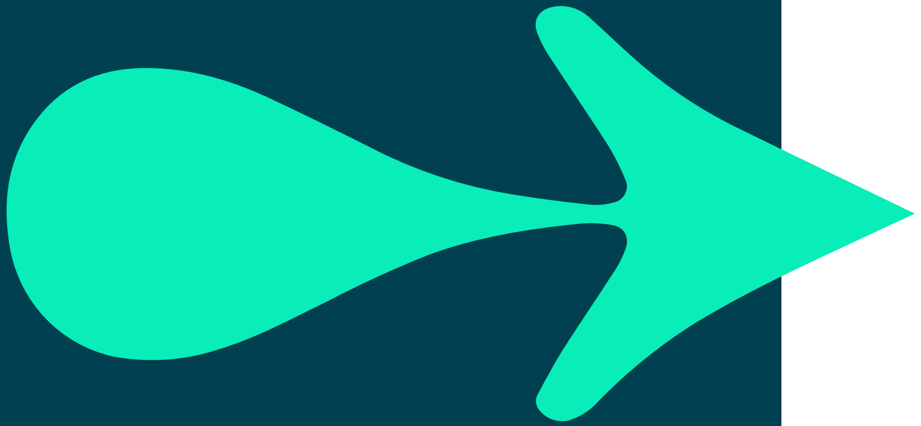
INFRASTRUCTURE AS CODE

Infrastructure as Code: An Essential DevOps Practice



IAC DEVOPS BEST PRACTICES

Configurations must be tested and monitored.



IAC DEVOPS BEST PRACTICES

Don't start out by automating everything.

IAC DEVOPS BEST PRACTICES

When introducing IAC tools to newcomers, tread cautiously.

IAC DEVOPS BEST PRACTICES

Include the developers in the creation of the IaC specifications for the runtime environments and infrastructure components.

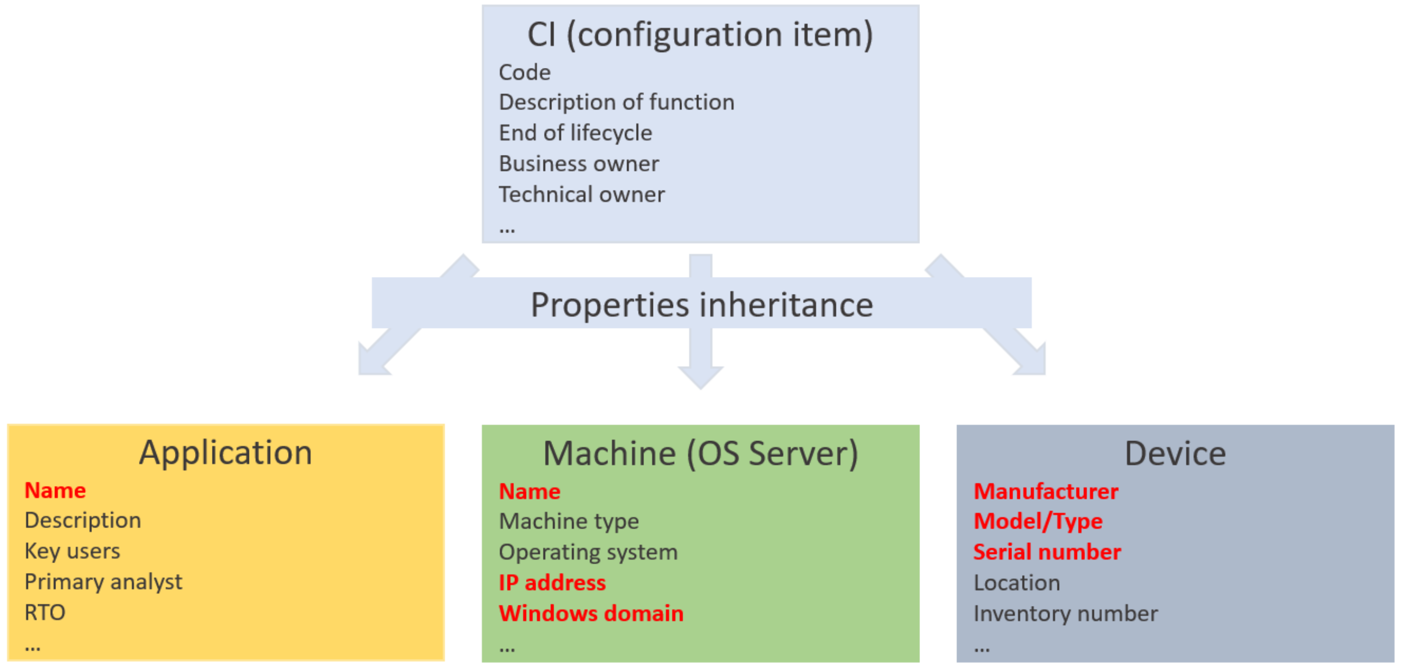
CONFIGURATION AS CODE

- A programme management technique known as "configuration as code" promotes the definition of configuration settings (such as environmental settings, resource provisioning, etc.) in code.

- This entails committing your software configuration settings to a version control repository and handling them the same way you would the rest of your code.

- As opposed to needing to generate and customise configuration for each deployment, or possibly having your configuration elsewhere outside of the repository.

CONFIGURATION AS CODE



PRACTICAL USE OF CONFIGURATION AS CODE

Utilising a specific type of control repositories for configuration.

PRACTICAL USE OF CONFIGURATION AS CODE

Creating a build and deployment procedure that is customised.

PRACTICAL USE OF CONFIGURATION AS CODE

Creating test environments that are configuration-specific.

PRACTICAL USE OF CONFIGURATION AS CODE

Secret management inside configurations.

PRACTICAL USE OF CONFIGURATION AS CODE

Confirming the provision of approval and quality control procedures.

POWERING
POTENTIAL

THANK YOU

Hope you enjoyed this learning journey.