DEVELOPMENT INFRASTRUCTURE

DEVELOPMENT/BUILD/COLLABORATION TOOLS

LECTURE 5- SOFTWARE ENGINEERING COURSE (LTAT.05.003)
DEVELOPMENT INFRASTRUCTURE & COLLABORATION

- Development infrastructure set the stage for development of the product.
- Less error prone
- Easier to include new team members
- Easier to follow the progress
TOPICS-TOOLS-TECHNOLOGIES

- Automation - Gradle
- VCS
- Branching
- VCS hosting
- Issue tracking
- Pull requests
- Wiki
SOFTWARE PROCESS AUTOMATION

**WHY**
- Software projects include many repetitive processes
- Often need to pass arguments to commands
- Doing so by hand is error-prone
- Hard to remember
- Repetitive
- Boring

**WHAT**
- Building your project
- Deploying to a server
- Generating documentation
- Running tests
- Managing dependencies
SOFTWARE PROCESS AUTOMATION

HOW

• Build scripts
  • Mini programs that describe the actions you wish to automate
  • Meant for a specific build script running software
• Different languages and formats for different development environments
• For Java projects, Gradle is often used
BUILD AUTOMATION TOOL

Gradle
GRADLE

• Gradle is a general-purpose build tool

• The core model is based on tasks

• Gradle has several fixed build phases

Gradle Documentation -> https://docs.gradle.org/current/userguide/what_is_gradle.html
The Java plugin adds Java compilation along with testing and bundling capabilities to a project. To use the Java plugin, include the following in your build script:

```
apply plugin: 'java'
```

**OR**

```
plugins {
    id 'java'
}
```

It defines tasks for a standard Java project
- Tests
- Building
- Packaging

The plugin assumes a standard Maven-inspired directory structure

For more info, [https://docs.gradle.org/current/userguide/java_plugin.html](https://docs.gradle.org/current/userguide/java_plugin.html)
RUNNING GRADLE TASKS
GRADLE - SUMMARY

• Build automation is an essential part of modern software development
• Each language/environment has their own tools
• Get comfortable with yours to make your work more efficient
• Google for plugins/dependencies before inventing the wheel
VERSION CONTROL SYSTEM (VCS)
WHAT IS A VCS

• A software tool that helps you keep track of changes in your data (code) over time.
  • SVN, Mercurial, Git, ...

• In a team, project changes in different ways, at different times, on different computers, and you are doing the same set of activities.
Git was released in 2005. It is …

- A distributed version control system
- Free and open-source (written primarily in C)
- Cross-platform
- Can be used for huge complicated projects with many collaborators
- Widely supported by popular tools, IDEs and repository hosting providers such as Bitbucket, Github, Gitlab etc.
- GUI clients
Git is used to manage repositories. The whole history of your repository is stored on your computer.

You can synchronize the history with other remote repositories by sending (pushing) your changes to them or downloading changes (pulling).
• A repository is a database of versions (commits or revisions).
• You can check out a commit into your working directory to work on it.
• You can commit your changes into the repository, creating a new commit.
• Each commit has a parent commit that came before it.

POSSIBLE STATES OF A FILE IN GIT

- Unmodified - just like it is in the repository.
- Modified - something about the file has been changed by you
- Staged - something has been changed and you want to commit the changes
- Committed - the data is safely stored in your local database

WHAT INFORMATION IS IN ONE COMMIT?

• Commit hash
  • Uniquely identifies a commit. Calculated from other contents of the commit and the hash of the parent
• Author
  • Who?
• Timestamp
  • When?
• Commit message
  • Why, what?
• Changes
  • What exactly?
• Parent commit(s)
HOW TO GET STARTED WITH GIT COMMAND LINE

https://git-scm.com/

• Linux - mostly pre-installed
  • apt-get install git-core

• mac OS - not pre-installed
  • Run git --version in your Terminal application to launch wizard

• Windows - not pre-installed
  • Download from Git website
USING GIT - GETTING REPOSITORIES

You typically obtain a Git repository in one of two ways:

• You can take a local directory that is currently not under version control, and turn it into a Git repository.

OR

• You can clone an existing Git repository from elsewhere.
USING GIT - GETTING REPOSITORIES

Initializing a Repository in an Existing Directory

$ cd <Enter path to folder>

$ git init

Initialized empty Git repository in C:/Users/anwar/Documents/SE course 2021/Test-Project/.git/

$ git remote add origin <url>
USING GIT - GETTING REPOSITORIES

For new/existing files type:

$ notepad test.txt  (add some text)
$ git add .
$ git commit -m 'Initial project version'

1 file changed, 1 insertion(+)
create mode 100644 test.txt
COMMON GIT COMMANDS

$ ls -lah (What’s inside the cloned repository)

total 17K
-drwxr-xr-x 1 DOMENIS+anwar 4096 0 Sep 27 15:14 ./
-drwxr-xr-x 1 DOMENIS+anwar 4096 0 Sep 27 14:52 ../
-drwxr-xr-x 1 DOMENIS+anwar 4096 0 Sep 27 15:15 .git/
-rw-r--r-- 1 DOMENIS+anwar 4096 34 Sep 27 15:14 test.txt
COMMON GIT COMMANDS

$ git log (for inspecting History)

commit b922c4172d2d6878d4d684264bf0311b6b28c3bb (HEAD -> master)
Author: Hina Anwar <anwar@ut.ee>
Date: Mon Sep 27 15:15:23 2021 +0300
Initial project commit
COMMON GIT COMMANDS

$ git status

On branch master
nothing to commit, working tree clean

• ‘Status’ is possibly the most important command in Git.
• Crucial to knowing what is the state of your local repository and working copy.
• Run it as much as possible to avoid errors and confusion.
STATUS AFTER CREATING A NEW FILE

$ notepad anothertest.txt
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  anothertest.txt

nothing added to commit but untracked files present (use "git add" to track)
USING GIT - CREATING COMMITS

• With the new file added, let’s make a commit

$ git commit

On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

anotherTest.txt

nothing added to commit but untracked files present (use "git add" to track)
USING GIT - ADDING A CHANGE

$ git add anothertest.txt
$ git status

On branch master
Changes to be committed:

( use "git restore --staged <file>..." to unstage)

new file: anothertest.txt
USING GIT - CREATING COMMITS

$ git commit
[master 5f18elf] adding another text file
1 file changed, 1 insertion(+)
create mode 100644 anothertest.txt
COMMITS ARE LOCAL

By default, Git doesn’t send your commits to other repositories

- $ git status

  On branch master
  
  Your branch is ahead of 'origin/master' by 1 commit.
  
  (use "git push" to publish your local commits)
  
  nothing to commit, working tree clean
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 78.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://bitbucket.org/hinaanwar2003/test_project.git
  e7559a0..00ceae7  master -> master
UPDATING YOUR REPOSITORY

$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 1.71 KiB | 103.00 KiB/s, done.
From https://bitbucket.org/hinaanwar2003/test_project
  * [new branch] master -> origin/master
BASIC COMMANDS

• These commands you will use 80% of the time
• Enough for managing a 1-person project and reap the benefits of Git

```
git clone
 git pull
 git add
 git commit
 git push
 git log
```
HOW TO EFFECTIVELY MANAGE A TEAM PROJECT

• Ideally, different people work on different features
• Need to separate your work from that of others
• Need to be able to only add your work when it’s done
BRANCHING

- To facilitate collaboration and ensure consistent practice, companies often enforce a branching model.
- A branch is where the history of your project diverges in two or more directions.
- In Git, branches have names for ease of use.
- The main branch, created by default, is usually named master.

Diagram:
- Master branch
- Jacks branch
- Marys branch
MERGING BRANCHES

• When a feature on a branch is ready, it can be merged into another branch, uniting the histories and code.
• After this, the temporary branch can be deleted
BRANCHING WORKFLOWS OR MODELS

• GitHub flow
• Git flow
• GitLab flow
GITHUB FLOW

• **Master** always contains stable state
• **Feature** are created from master for new features
• **Feature** branches are merged into **Master** when feature has been tested
GIT FLOW

- **Master** branch always contains stable release ready state of code.
- **Develop** branch always contains stable state.
- **Feature** branches are created from Develop branch.
- Feature branches are merged into develop when feature is tested.
- Develop branch is merged into Master when it’s time for a new release.
**GIT FLOW - CONT’D..**

- **Hotfix** branches are created from Master for time critical, small fixes.
- Merged into both **Master** and **Develop** when done.
GIT FLOW – CONT’D..

• Release branches are created from Develop when it’s time to prepare for release
• Used for final tests of feature-set
• Can commit bug-fixes to them
• When ready for release, merge into Master and Develop
• Helps to isolate release from ongoing work
GITLAB FLOW

• Similar to GitHub flow i.e., it also uses Master and Feature branches
• Additional branches for environment and release
USING GIT - BRANCHES

$ git status
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean

$ git branch  (List Branches)

$ git branch new-branch  (Adding Branches)
CHANGING BRANCHES

$ git checkout new-branch
Switched to branch 'new-branch'

$ git status
On branch new-branch
nothing to commit, working tree clean

$ git branch
  master

  * new-branch
REMEMBER

• Git will not do many branch-related things for you:
  • Commits on a branch will not be available on other branches until you merge
  • Branches other than the active one will not be pushed
• Be mindful of where you are and what you’re doing
MERGING BRANCHES

$ git status
On branch master
nothing to commit, working tree clean

$ git merge new-branch
Updating f526ea5..5d46150
Fast-forward
  a.txt | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 a.txt
MERGE CONFLICTS

• In a perfect world, that would be it
• No real project is perfect
• Git gets confused when the same parts of the files are changed in different ways on different branches
Content of text file

an old silent pond
a frog jumps into the pond
splash! silence again

An old silent pond
A frog jumps into the pond
splash! Silence again

Which changes should be used?

Changes made on Master branch

Changes made on new branch
MERGE CONFLICT

$ git merge new-branch
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
STATUS IN CONFLICTED STATE

$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified: test.txt
no changes added to commit (use "git add" and/or "git commit -a")
TEST.TXT IN CONFLICTED STATE

<<<<<<< HEAD
an old silent pond...

a frog jumps into the pond,
splash! silence again.

=======

An old silent pond
A frog jumps into the pond
splash! Silence again

>>>>>>> new

Resolving a merge conflict
• Find conflicts in file.
• Decide what is the desired final state.
• Delete the conflict markers, leaving only the desired state.
• Add the resolved files and commit.
an old silent pond...
A frog jumps into the pond,
splash! silence again.

An old silent pond
A frog jumps into the pond
splash! Silence again

new-branch
an old silent pond...
a frog jumps into the pond,
splash! silence again.

An old silent pond
A frog jumps into the pond
splash! Silence again

<<<<<<< HEAD
an old silent pond...
a frog jumps into the pond,
splash! silence again.
=======
An old silent pond
A frog jumps into the pond
splash! Silence again

Changes on current branch

>>>>>> new-branch
an old silent pond...
a frog jumps into the pond,
splash! silence again.

An old silent pond
A frog jumps into the pond
splash! Silence again

Changes on the branch that we merge
An old silent pond…
A frog jumps into the pond,
splash! Silence again.
AFTER SOLVING MERGE CONFLICT

$ git add test.txt
$ git commit
[master 4b1bee1] Merge branch 'new-branch'
BRANCHING COMMANDS

• This covers probably 95% of typical Git usage
• There is much more
  • Aliases
  • Rebasing
  • Cherry-picking
  • Bisecting
  • Git hooks
  • Traversing history and undoing changes

```
git branch

git branch branch-name

git checkout

git merge
```
RECAP

• A VCS like Git helps you track changes in your project and collaborate
• Git is the most popular modern VCS enabling powerful collaboration in big projects
• Git is “stupid” and doesn’t do many things for you
• Use git status a lot, be aware of where you are, and what you’re doing
ISSUE TRACKING, COMMITS, PULL REQUEST

BITBUCKET

Lecture 5- Software engineering course (LTAT.05.003)
REPOSITORY HOSTING

• Typically, a repository is backed up to a server
• Considered the source of truth
• All changes are sent there, and all team members get updates from there
• Often have additional features
  • More comfortable history viewing
  • Automation, CI/CD (Next lecture!)
  • Documentation
  • Issue tracking
BITBUCKET ISSUE TRACKER

• Needs to be enabled in repository settings
• A relatively simple issue tracking solution
• Tightly integrated with your Git repository
• Can be made public or private
• In a public tracker, anyone can report issues
  • For open-source projects
ANATOMY OF AN ISSUE

• Title - for quick overview
• Description - detailed description of work to be done
• Assignee - who is expected to work on the issue
• Kind - separate bugs from feature requests
• Priority - help developers pick more crucial tasks
• Component (optional) - separate issues by areas of the project
• Milestone (optional) - during what time should the issue be completed
• Version (optional) - in what version of the project is the issue completed
BITBUCKET ISSUE

Hina Anwar / sample_project_testing.git / test_project

Issues

Create issue

Title*

Description

Normal text B I ...

Assignee

Find a user by their name.. Assign to me

Kind*

bug

Priority*

major

Attachments

Select files

Create issue Cancel
BITBUCKET ISSUE

Hina Anwar / sample_project_testing.git / test_project / Issues

Update Readme.txt

Issue #1  NEW

Hina Anwar  REPO OWNER created an issue 52 seconds ago
Add more text into this file

Comments (0)

HA

What would you like to say?

Assignee  Hina Anwar
Type  task
Priority  minor
Status  new
Votes  0  Vote for this issue
Watchers  1  Stop watching
LIFE-CYCLE OF AN ISSUE

• Issue tracking software typically defines several possible states of an issue
• Bitbucket has some default states
• Open → In progress → Resolved/Cancelled
INTEGRATING ISSUES WITH COMMITS

- Often, issue tracking software lets you integrate issues with your Git repository
- This way, you can easily see what commits relate to what issues
- For this, include a specific syntax in your message:

\[
\text{
$ git commit -m "your message, see #1"$
}
\]

When we include an issue number in the commit message we can do several things, for example:

- Refer to an issue with "see #<issue number>"
- Close an issue with "close #<issue number>"

https://confluence.atlassian.com/bitbucket/resolve-issues-automatically-when-users-push-code-221451126.html
PULL REQUESTS (PR)

• Signal to maintainer of the repository -> “Please pull my changes (branch) into the repository”

• Sometimes called a merge request -> “Please merge my changes into the given branch”

• Gives a maintainer overview of what is being done

• One PR per one branch

• Crucial part of collaboration in modern software projects
COLLABORATING ON PROJECTS - FULL LIFE-CYCLE OF A FEATURE

- Reporting an issue
- Assigning an issue
- Creating a new feature branch for the issue
- Working on the issue
- Creating a Pull Request for the issue
- Waiting for approvals
- Merging the Pull Request
BITBUCKET WIKI

• A place to document your project
• Can create many pages, link them together
• Uses Markdown for rich text styling
• Is contained in a VCS repository, so you can see history and roll back changes
• Needs to be enabled in repository settings
Welcome

Welcome to your wiki! This is the default page we've installed for your convenience. Go ahead and edit it.

Wiki features

This wiki uses the Markdown syntax. The MarkDownDemo tutorial shows how various elements are rendered. The Bitbucket documentation has more information about using a wiki.

The wiki itself is actually a git repository, which means you can clone it, edit it locally/offline, add images or any other file type, and push it back to us. It will be live immediately.

Go ahead and try:

```
$ git clone https://bitbucket.org/hines/herbology2003/test_project.git
```

Wiki pages are normal files, with the .md extension. You can edit them locally, as well as creating new ones.

Syntax highlighting

You can also highlight snippets of text (we use the excellent Pygments library).

Here's an example of some Python code:

```python
def wiki_respects(text):
    formatter = lambda t: "funky" + t
    return formatter(text)
```

You can check out the source of this page to see how that's done, and make sure to bookmark the vast library of Pygment lexers, we accept the 'short name' or the 'mimetype' of anything in there.

Have fun!
TOPICS COVERED

• Automation, Gradle
• VCS, Git, branching
• Branching models
• Bitbucket issues, pull requests
• Collaboration flows
THANK YOU