CMPT 332 - Operating Systems Concepts
Git Introduction (Part 2)

1 Branching

Git branches allow you to work concurrently with other team members. Branches are simply pointers to commits. The chain of commits a repository may have can branch out into a tree-like structure. In fact, there is no actual storage for branches other than a table with branch names and corresponding commit hashes.

To learn more about the details of how Git implements and manages branches, you can read the following article. `https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell`

```
# create a branch
$ git branch <branch_name>

# switch to the branch
$ git checkout <branch_name>

# or alternatively, you can create and switch branches in one step
$ git checkout -b <branch_name>

# you can also push the branch to remote server so that your partner can checkout
# the branch
$ git push -u origin <branch_name>

# list local branches and see which branch you're on
$ git branch

# list all branches (local and remote)
$ git branch -a
```

There are many branching models that can be used depending on the development environment and the needs of the team. Gitflow is a popular branching model that has gained popularity `https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`. We recommend following a similar model for the assignments. You can create a main assignment branch for every assignment (e.g. a1, a2, ...) and treat it as a development branch. Every team member working on a feature can branch out from the assignment branch and implement the intended feature. Consider the following example:

```
# starting from the master branch
$ git checkout -b a1
$ git push -u origin a1

# create 2 feature branches where you and your partner implement the system call and
# user program for assignment 1 part A
$ git branch feature/howmanycmpt-syscall
$ git branch feature/howmanycmpt-user-test
```

This allows you to work independently from other team members. Once a feature is done, you can merge the feature branch to the assignment "development" branch. You can also check out other branching models such as trunk-based workflows.

2 Merging Branches

Now you're done with a feature branch and you want to merge that feature to your development branch. You do so using the following:

```
# go to a1
$ git checkout a1

# merge feature branch to branch 'a1'
$ git merge feature/howmanycmpt-syscall

# update the remote a1 branch with the latest merged state
$ git push
```

3 Resolving Conflicts

Merging branches can sometimes lead to conflicts. Consider the case where you and your partner have both modified the same line in the same file. For instance, in our example both you and your partner may edit the file user/user.h to add a user-level header for the new system call. The first merge will go smoothly. When it is time to merge the other branch, however, git will detect a conflict and the merge will be aborted. To resolve the conflict you will need to edit the conflicting files and choose which change you want to keep. You can use *git status* to see which files have conflicts. Once you have fixed the conflicts, you can simply use *git add* followed by *git commit* to continue the aborted merge operation.

Example

```
# initialize a local repo
$ mkdir test
$ cd test
$ git init

# make an initial commit on the master branch
$ echo "this is the master branch" > readme.txt
$ git add .
$ git commit -m "initial commit"
[master (root-commit) 05b0503] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt

# branch out and make change
$ git checkout -b feature/a
Switched to a new branch 'feature/a'
$ echo "this is feature a" > readme.txt
$ git add .
$ git commit -m "implement feature a"
```

```
[feature/a 3b13b19] implement feature a
 1 file changed, 1 insertion(+), 1 deletion(-)

# go back to the master branch
$ git checkout master

# branch out and make another change
$ git checkout -b feature/b
Switched to a new branch 'feature/b'
$ echo "this is feature b" > readme.txt
$ git add .
$ git commit -m "implement feature b"
[feature/b b3f629b] implement feature b
 1 file changed, 1 insertion(+), 1 deletion(-)

# go back to the master branch
$ git checkout master

# merge feature a
$ git merge feature/a
Updating 05b0503..3b13b19
Fast-forward
 readme.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

# merge feature b
$ git merge feature/b
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

$ 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:  readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

At this point merging feature b is aborted. Edit the conflicting file. You should see something that looks like the following:

```
<<<<<<< HEAD
this is feature a
=======
this is feature b
>>>>>>> feature/b
```

Fix the conflicting file, keeping either one of the changes or keeping both changes if this is needed. Finally, you can continue the merge using the following:

```
$ git add .
```

```
$ git commit
[master 3df8b23] Merge branch 'feature/b'
```

The log at this point should resemble the following:

```
$ git log
commit 3df8b237c3f5820e9ee0a63e363afb74e954afa1 (HEAD -> master)
Merge: 3b13b19 b3f629b
Author: John Doe <john.doe@example.com>
Date:  Sun Sep 19 12:20:59 2021 -0600

    Merge branch 'feature/b'

commit b3f629bd6d4ecf27c3733ccacb70cdf962f89b81 (feature/b)
Author: John Doe <john.doe@example.com>
Date:  Sun Sep 19 12:14:52 2021 -0600

    implement feature b

commit 3b13b193fab2ec0f846769b41fef8a03d916f618 (feature/a)
Author: John Doe <john.doe@example.com>
Date:  Sun Sep 19 12:11:45 2021 -0600

    implement feature a

commit 05b0503def3a9a596be22c50b4cd39f5cc9dd26c
Author: John Doe <john.doe@example.com>
Date:  Sun Sep 19 12:08:10 2021 -0600

    initial commit
```

4 More Git

Git is a complex and powerful tool and there is always more to learn. However, Git is not the primary focus of this course. We have covered a sufficient introduction that allows you to perform the basic operations you will most likely need during this course, but this is far from everything that Git can do.

`https://git-scm.com/docs` is great for documentation. If you are wondering about some detail or trying to figure out how to do something, the documentation is a good place to start. `https://learngitbranching.js.org/` is an interactive learning tool that focuses on branching, merging, rebasing, and cherry-picking.