1. Hello

1.1. Git-Katas

1.1.1. Basic Commits

1. Use `git status` to see which branch you are on.

```
â¯ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

2. What does `git log` look like?

```
â¯ git log
fatal: your current branch 'master' does not have any commits yet
```

3. Create a file.

```
â¯ touch file
```

4. What does the output from `git status` look like now?

```
â¯ git status
On branch master

No commits yet

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

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

5. `add` the file to the staging area.

```
â¯ git add file
```

6. How does `git status` look now?

```
â¯ git status
On branch master

No commits yet

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

7. `commit` the file to the repository.

```
â¯ git commit -m "Commited 'file'."
[master (root-commit) 0dbee8a] Commited 'file'.
 1 file changed, 0 insertions(+), 0 deletions(-)
```

```
 create mode 100644 file
```

8. How does `git status` look now?

```
â¯ git status
On branch master
nothing to commit, working tree clean
```

9. Change the content of the file you created earlier.

```
â¯ echo "hello" > file
```

10. What does `git status` look like now?

```
â¯ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file

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

11. `add` the file change.

```
â¯ git add file
```

12. What does `git status` look like now?

```
â¯ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file
```

13. Change the file again.

```
â¯ echo "again" >> file
```

14. Make a `commit`.

```
â¯ git commit -m "Changed 'file'."
[master 723a20a] Changed 'file'.
 1 file changed, 1 insertion(+)
```

15. What does the `status` look like now? The `log`?

```
â¯ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file

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

â¯ git log
commit 723a20aa06fb708c48da6d49ab99f440995395e5 (HEAD -> master)
Author: Felix Friesenbichler <felixfriesenbichler@protonmail.com>
Date:   Wed Sep 18 10:42:25 2024 +0200

    Changed 'file'.

commit 0dbee8a5174d0dce5405c03b17bcff335fc2d03c
Author: Felix Friesenbichler <felixfriesenbichler@protonmail.com>
Date:   Wed Sep 18 10:37:48 2024 +0200
```

```
    Commited 'file'.
```

16. Add and commit the newest change.

```
â¯ git add file
â¯ git commit -m "Changed 'file' again."
[master bebdecb] Changed 'file' again.
 1 file changed, 1 insertion(+)
```

1.1.2. Basic Staging

1. What's the content of `file.txt`?

```
cat file.txt
```

2. Overwrite the content in `file.txt`.

```
echo 2 > file.txt
```

3. What does `git diff` tell you?

```
â¯ git diff
diff --git a/file.txt b/file.txt
index d00491f..0cfbf08 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1 @@
-1
+2
```

4. What does `git diff -staged` tell you? Why is this blank?

Since nothing is staged, there is no output.

5. Run `git add file.txt` to stage your changes from the working directory.

```
git add file.txt
```

6. What does `git diff` tell you?

Since nothing is unstaged, there is no output.

7. What does `git diff --staged` tell you?

```
diff --git a/file.txt b/file.txt
index d00491f..0cfbf08 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1 @@
-1
+2
```

8. Overwrite the content in `file.txt` to cahnge the state of your file in the working directory.

```
echo 3 > file.txt
```

9. What does `git diff` tell you?

```
diff --git a/file.txt b/file.txt
index 0cfbf08..00750ed 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1 @@
-2
+3
```

10. What does `git diff --staged` tell you?

```
diff --git a/file.txt b/file.txt
index d00491f..0cfbf08 100644
```

```
--- a/file.txt
+++ b/file.txt
@@ -1 +1 @@
-1
+2
```

11. Explain what is happening.

The `git diff` from ‘9’ shows the difference between what is staged and what is changed, while ‘10’ highlights the difference between what is staged and what originally stood in the file.

12. Run `git status` and observe that `file.txt` are present twice in the output.

```
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file.txt
```

13. Run `git restore --staged file.txt` to unstage the change.

```
â¯ git restore --staged file.txt
```

14. What does `git status` tell you now?

```
â¯ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file.txt

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

15. Stage the change and make a commit.

```
â¯ git commit -m "Changed content of ’file.txt’"
[master d51b6e3] Changed content of ’file.txt’
 1 file changed, 1 insertion(+), 1 deletion(-)
```

16. What does the log look like?

```
â¯ git log
commit d51b6e32dd37fdce84e7d4ff95f8b2cb94851d5d (HEAD -> master)
Author: Felix Friesenbichler <felixfriesenbichler@protonmail.com>
Date:   Wed Sep 18 12:05:12 2024 +0200

    Changed content of ’file.txt’

commit 864bbd938efa4947bd9410cd1778517cc580274b
Author: git-katas trainer bot <git-katas@example.com>
Date:   Wed Sep 18 10:48:02 2024 +0200

    1
```

17. Overwrite the content in `file.txt`.

```
â¯ echo 4 > file.txt
```

18. What is the content of `file.txt`?

```
â¯ cat file.txt
4
```

19. What does `git status` tell us?

```
â¯ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file.txt

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

20. Run `git restore file.txt`.

```
â¯ git restore file.txt
```

21. What is the content of `file.txt`?

```
â¯ cat file.txt
3
```

22. What does `git status` tell us?

```
â¯ git status
On branch master
nothing to commit, working tree clean
```

1.1.3. Basic Branching

1. Use `git branch` to see the two branches that are relevant for this exercise.

```
â¯ git branch
* master
  second-branch
```

2. What branch are you on?

I am currently on branch `master`.

3. Use `git branch mybranch` to create nwe branch called ‘mybranch’.

```
â¯ git branch mybranch
```

4. Use `git branch` again to see the new branch created.

```
â¯ git branch
* master
  mybranch
  second-branch
```

5. Use `git switch mybranch` to switch to your new branch.

```
â¯ git switch mybranch
Switched to branch 'mybranch'
```

6. How does the output from `git status` change when you switch between the ‘master’ and the enw branch that you have created?

```
â¯ git status
On branch mybranch
nothing to commit, working tree clean
â¯ git switch master
Switched to branch 'master'
â¯ git status
On branch master
```

```
nothing to commit, working tree clean
```

7. How does the workspace change when you change between the two branches?

The workspace does not change at all.

8. Make sure you are on your ‘mybranch’ branch before you continue.

```
git switch mybranch
```

9. Create a file called `file1.txt` with your name.

```
â¯ touch file1.txt; echo "felix" > file1.txt
```

10. `Add` the file and `commit` with this change.

```
â¯ git add file1.txt
â¯ git commit -m "Added file ’file1.txt’"
[mybranch 7a5bfff] Added file ’file1.txt’
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

11. Use `git log --oneline --graph` to see your branch pointing to the new commit.

```
â¯ git log --oneline --graph
* 7a5bfff (HEAD -> mybranch) Added file ’file1.txt’
* a4c46e5 (second-branch, master) dummy commit
```

12. Switch back to the branch called ‘master’.

```
â¯ git switch master
Switched to branch ’master’
```

13. Use `git log --oneline --graph` and notice how the commit you made on the ‘mybranch’ branch is missing on the ‘master’ branch.

```
â¯ git log --oneline --graph
* a4c46e5 (HEAD -> master, second-branch) dummy commit
```

14. Make a new file called `file2.txt` and commit that file.

```
â¯ touch file2.txt
â¯ git add file2.txt
â¯ git commit -m "Added file ’file2.txt’"
[master 0e51651] Added file ’file2.txt’
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2.txt
```

15. Use `git log --oneline --graph --all` to see your branch pointing to the new commit, and that the two branches now have different commits on them.

```
â¯ git log --oneline --graph --all
* 0e51651 (HEAD -> master) Added file ’file2.txt’
| * 7a5bfff (mybranch) Added file ’file1.txt’
|/
* a4c46e5 (second-branch) dummy commit
```

16. Switch to your branch ‘mybranch’.

```
â¯ git switch mybranch
Switched to branch ’mybranch’
```

17. What happened to your working directory? Can you see your ‘file2.txt’?

I cannot see the file ‘file2.txt’, because it only exists on the ‘master’ branch.

18. Use `git diff mybranch master` to see the difference between the two branches.

```
â¯ git diff mybranch master
diff --git a/file1.txt b/file1.txt
deleted file mode 100644
```

```
index b188d8f..0000000
--- a/file1.txt
+++ /dev/null
@@ -1 +0,0 @@
-felix
diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..e69de29
```

1.1.4. Fast-Forward Merge

1. Create a (feature)branch called `feature/uppercase`. (Yes, `feature/uppercase` is a perfectly legal branch anme, and a common convention.)

```
â¯ git branch feature/uppercase
```

2. Switch to this branch.

```
â¯ git switch feature/uppercase
Switched to branch 'feature/uppercase'
```

3. What is the output of `git status`?

```
â¯ git status
On branch feature/uppercase
nothing to commit, working tree clean
```

4. Edit the 'greeting.txt' to contain an uppercase greeting.

```
â¯ echo "HELLO" > greeting.txt
```

5. Add `greeting.txt` files to staging area and commit.

```
â¯ git commit -m "Changed greeting to uppercase in 'greeting.txt'"
[feature/uppercase 99af220] Changed greeting to uppercase in 'greeting.txt'
 1 file changed, 1 insertion(+), 1 deletion(-)
```

6. What is the output of `git branch`?

```
â¯ git branch
* feature/uppercase
  master
```

7. What is the output of `git log --oneline --graph --all`?

```
â¯ git log --oneline --graph --all
* 99af220 (HEAD -> feature/uppercase) Changed greeting to uppercase in 'greeting.txt'
* 59ab95c (master) Add content to greeting.txt
* 42953e4 Add file greeting.txt
```

8. Switch to the `master` branch.

```
â¯ git switch master
Switched to branch 'master'
```

9. Use `cat` to see the contents of the `greeting.txt`.

```
â¯ cat greeting.txt
hello
```

10. Diff the branches

```
â¯ git diff master feature/uppercase
diff --git a/greeting.txt b/greeting.txt
index ce01362..e427984 100644
--- a/greeting.txt
+++ b/greeting.txt
@@ -1 +1 @@
```

```
-hello
+HELLO
```

11. Merge the branches

```
â¯ git merge feature/uppercase
Updating 59ab95c..99af220
Fast-forward
 greeting.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

12. Use `cat` to see the contents of the greetings.

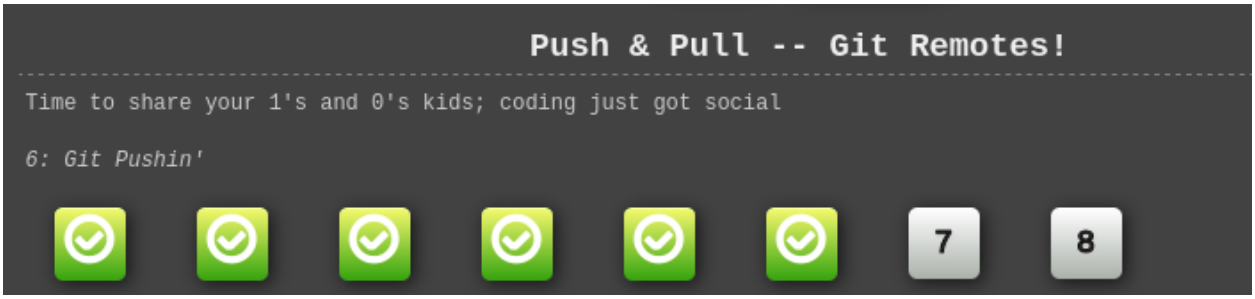
```
â¯ cat greeting.txt
HELLO
```

13. Delete the uppercase branch.

```
â¯ git branch -d feature/uppercase
Deleted branch feature/uppercase (was 99af220).
```

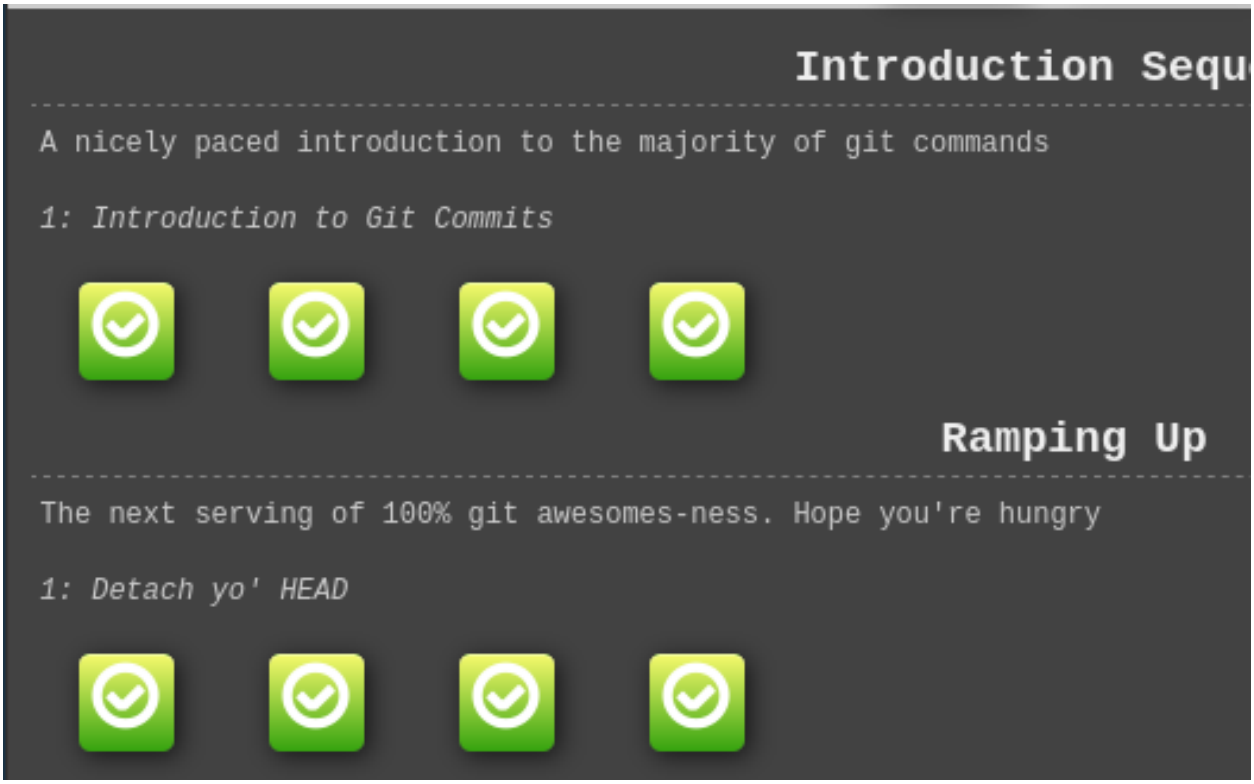
1.2. Git-Branching

1.2.1. Main



1.2.1.1. Introduction Sequence & Ramping Up

1.2.2. Remote


Introduction Sequ
A nicely paced introduction to the majority of git commands
1: Introduction to Git Commits
Ramping Up
The next serving of 100% git awesomes-ness. Hope you're hungry
1: Detach yo' HEAD