

Dr. Mindaugas Šarpis

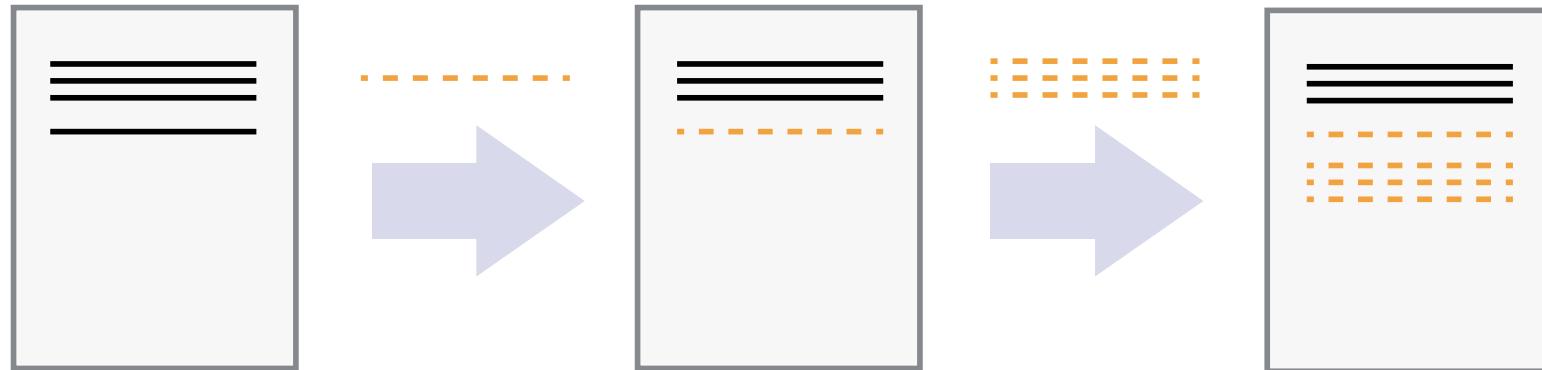
Lessons on Data Analysis from CERN

Lecture 8

Version Control using Git

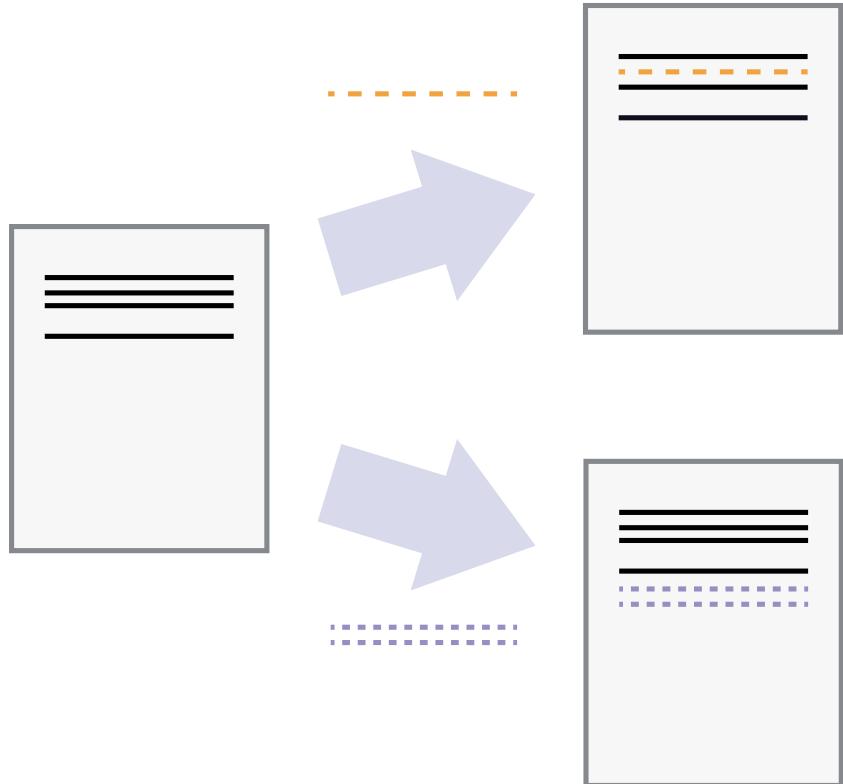
Tracking Changes (differences)

- Rather than saving multiple copies of the same file, we can track changes.
- Word processors and other software have some change-tracking functionality but it is limited (no synchronous editing, no change history, etc.).
- `git` is an open-source version control system that is used to track changes in files.



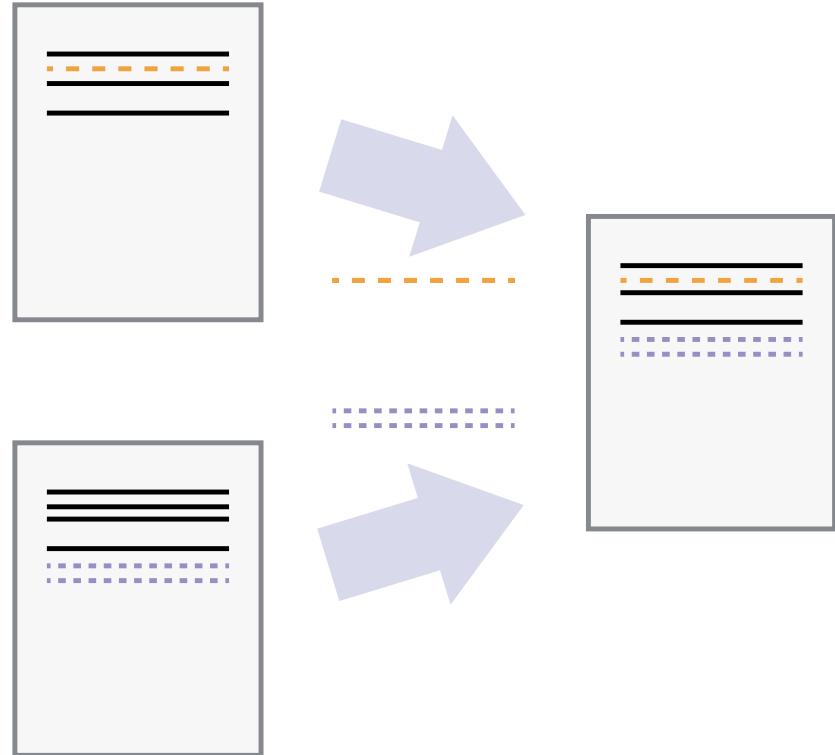
Different Versions

- An edit to a file might overwrite some of the content in the previous version.
- This *divergences* may arrise while working alone, but they are really common when multiple people are working on the same file.



Merging

- `git` has great functionality for merging different versions of the same file.
- If the previous content is not overwritten, or deleted, merge just combines the changes into one file.
- If changes over-write each other a so-called **merge conflict** arises.



Using git for the first time

- The user name and email address need to be configured.

```
git config --global user.name "Mindaugas Sarpis"  
git config --global user.email "mindaugas.sarpis@cern.ch"
```

- Check the configuration with:

```
git config --list
```

- Edit the configuration with:

```
git config --global --edit
```

- Open configuration help:

```
git config --h  
git config --help
```

usage: `git config [<options>]`

Config file location

| | |
|--------------------------------------|---------------------------------|
| <code>--global</code> | use global config file |
| <code>--system</code> | use system config file |
| <code>--local</code> | use repository config file |
| <code>--worktree</code> | use per-worktree config file |
| <code>-f, --file <file></code> | use given config file |
| <code>--blob <blob-id></code> | read config from given blob obj |

Action

| | |
|-------------------------------|-----------------------------------|
| <code>--get</code> | get value: name [value-pattern] |
| <code>--get-all</code> | get all values: key [value-patter |
| <code>--get-regexp</code> | get values for regexp: name-regex |
| <code>--get-urlmatch</code> | get value specific for the URL: s |
| <code>--replace-all</code> | replace all matching variables: n |
| <code>--add</code> | add a new variable: name value |
| <code>--unset</code> | remove a variable: name [value-pa |
| <code>--unset-all</code> | remove all matches: name [value-p |
| <code>--rename-section</code> | rename section: old-name new-name |
| <code>--remove-section</code> | remove a section: name |
| <code>-l, --list</code> | list all |
| <code>--fixed-value</code> | use string equality when comparin |
| <code>-e, --edit</code> | open an editor |
| <code>--get-color</code> | find the color configured: slot [|

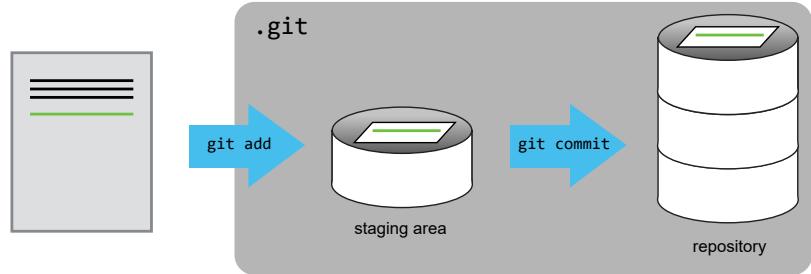
Creating a new repository

- A repository is initialized with the following command:

```
git init
```

- This command creates a new repository in the current directory.
- The repository is a hidden directory called `.git` that contains all the information changes tracked by `git`.
- You can check the status of the repository with:

```
git status
```



- The repository is empty at this point and the output will be:

```
On branch main
```

```
No commits yet
```

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

Staging Area

- `git` has a staging area where files are placed to track the changes made to them.
- To move a file to the staging area use:

```
git add <file>
```

- To move all files to the staging area use:

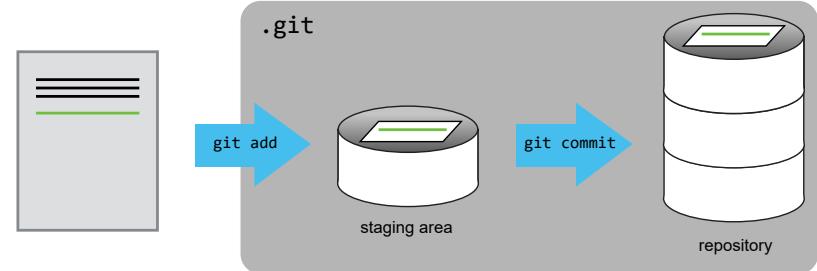
```
git add --all
```

- To unstage a file use:

```
git restore --staged <file>
```

- Changes to files can be viewed with:

```
git diff
```



- When staged files are present, the output of `git status` will be:

```
On branch main
Your branch is up to date with 'origin/main'.

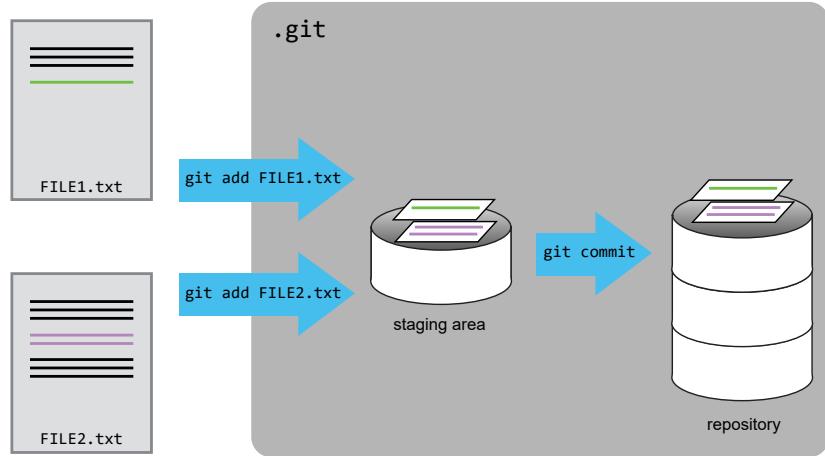
Changes to be committed:
  (use "git restore --staged <file> ..." to unstage)
    modified:   < file >
```

Committing Changes

- Files are committed to the repository from the staging area with:

```
git commit -m "A message describing the changes"
```

- Commit is a snapshot of the repository at a given time.
- Only changes to files are tracked, not the directories themselves.
- It's best to keep the commits small and focused on a single change.
- The commit message should be descriptive and concise.
- The commit message should be in the present tense.



Restoring Changes

- Changes to files can be restored to the last commit with:

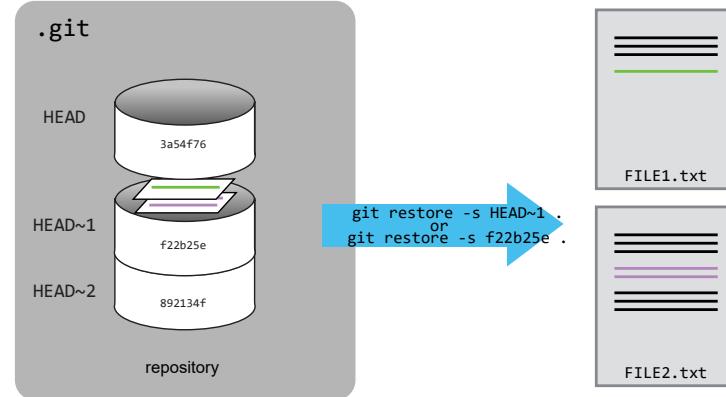
```
git restore < file >
```

- Changes to files can be restored to the last commit and the staging area with:

```
git restore --staged < file >
```

- Changes to files from previous commits can be restored using the *hash* of the commit:

```
git restore --source=<hash> < file >
```

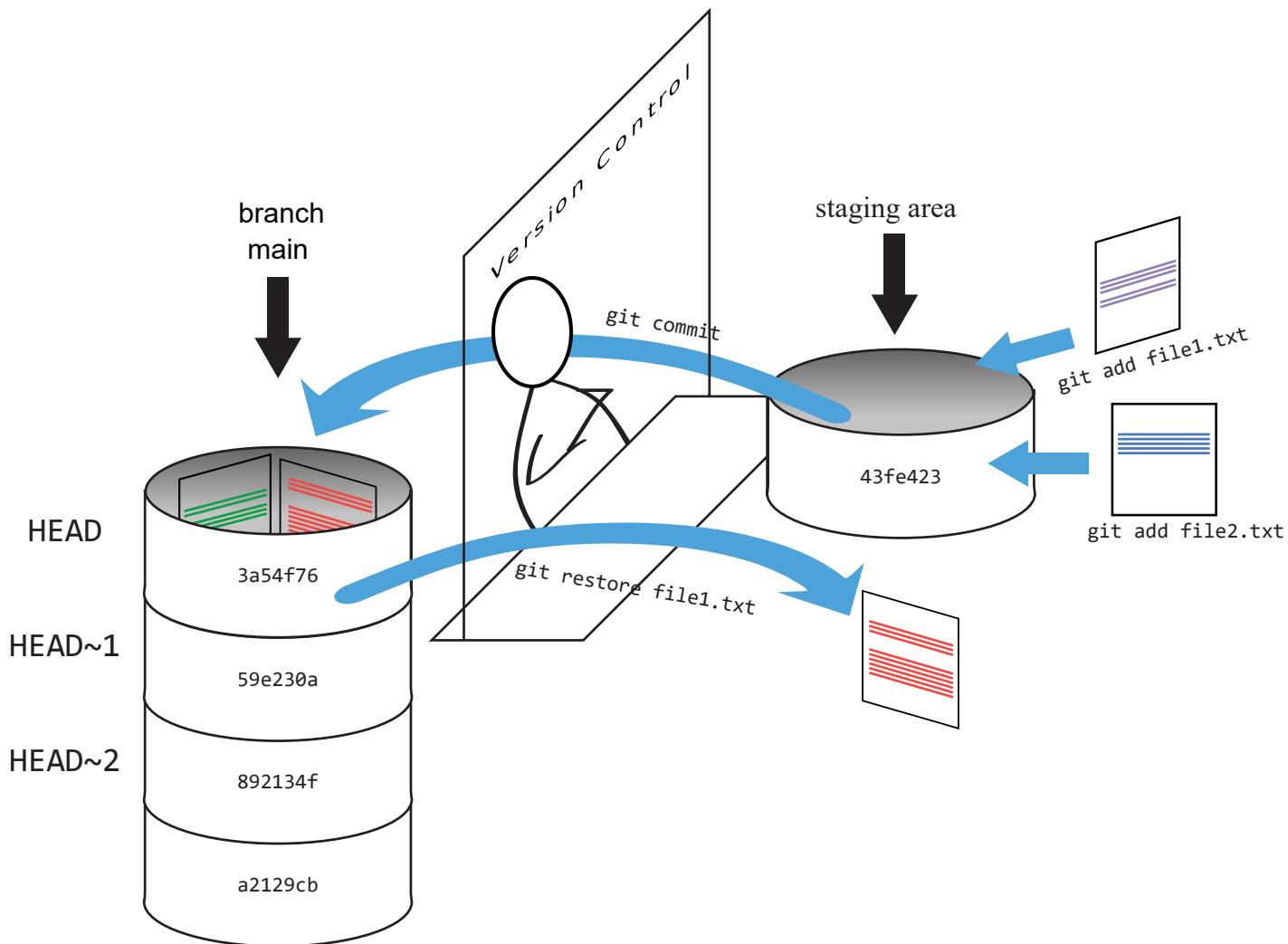


- A new commit reverting the changes can be made with:

```
git revert < hash >
```

- The entire repository can be restored to the last commit with deleting the changes:

```
git reset --hard < hash >
```



Ignoring Files and Directories

- There might be files that you don't want to track with `git`.
 - Temporary files
 - Output files
 - Files with sensitive information
 - Large files
- These files can be ignored by creating a `.gitignore` file in the repository.

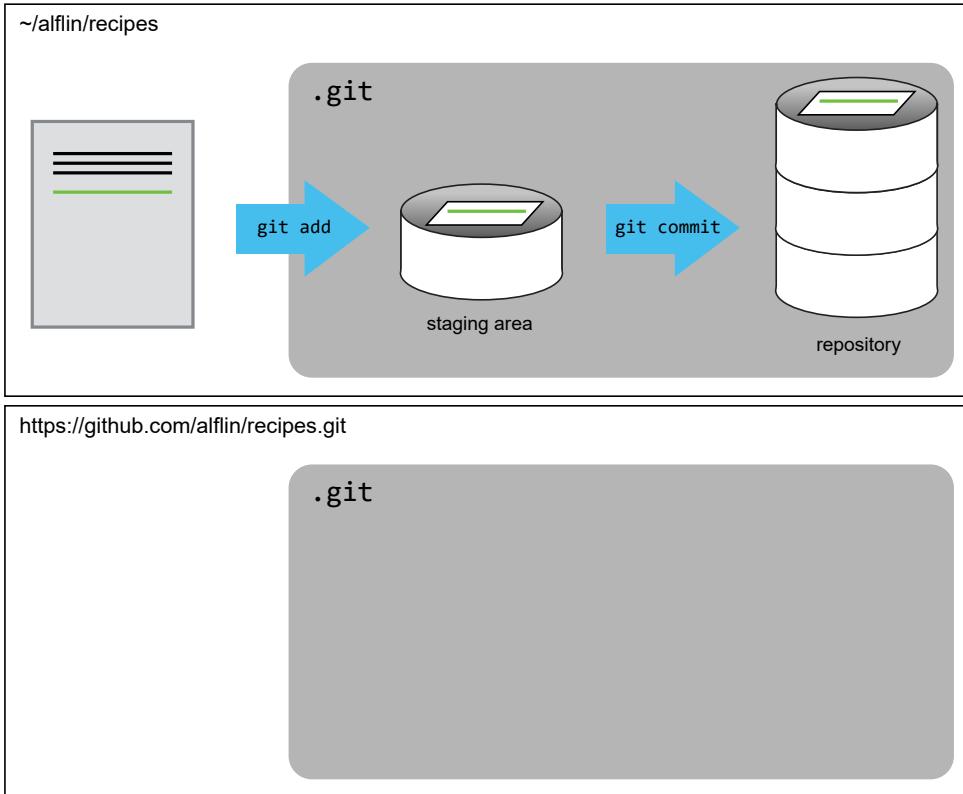
```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
```

Git Remotes

- One of the most powerful features of `git` is the ability to work with remote repositories.
- Remote repositories are copies of the repository that are stored on a server.
- Using one of the remote providers (GitHub, GitLab, Bitbucket, etc.) you can store your repository in the cloud.
- This enables collaboration with other people and provides a backup of your work.



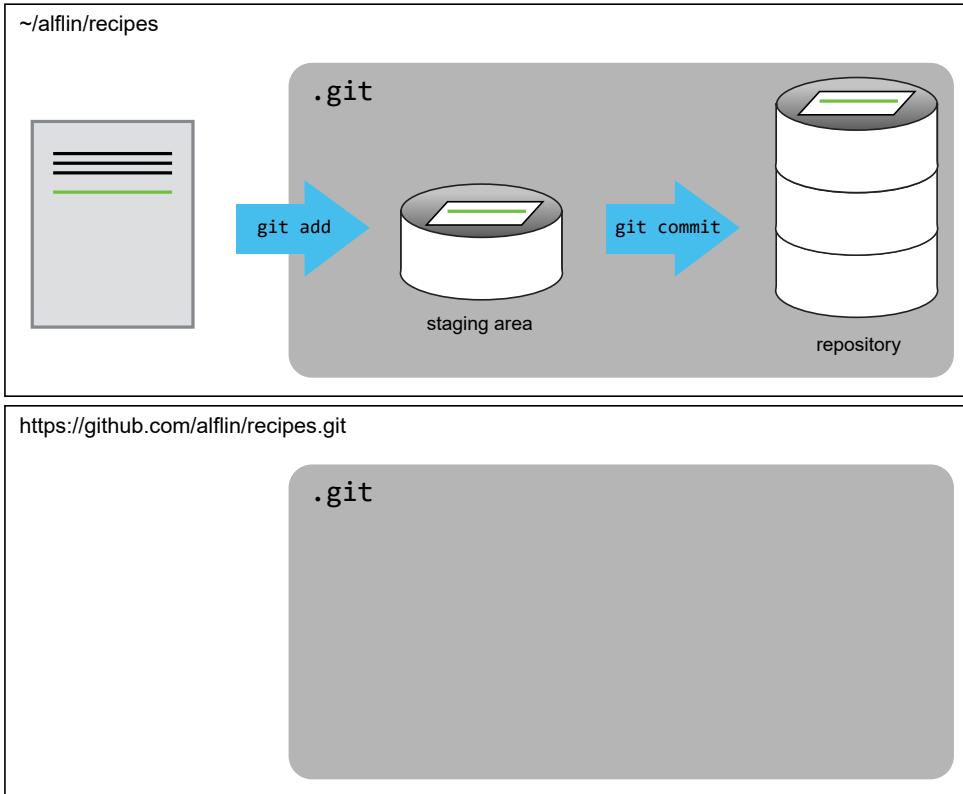
Git Remotes

- The remote is created via the remote provider (GitHub, GitLab, Bitbucket, etc.).
- A remote URL needs to be added to the local repository with:

```
git remote add origin git@github.com:mygithub/myrepo
```

- To check which remotes are added:

```
git remote -v
```



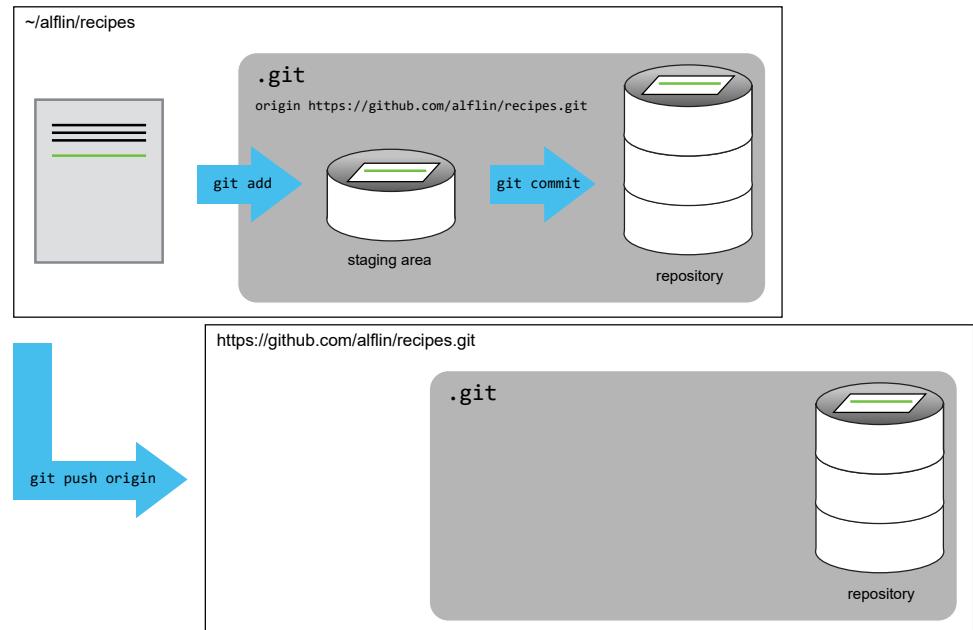
Push / Pull Operations

- Changes to the local repository can be pushed to the remote repository with:

```
git push origin main
```

- Changes to the remote repository can be pulled to the local repository with:

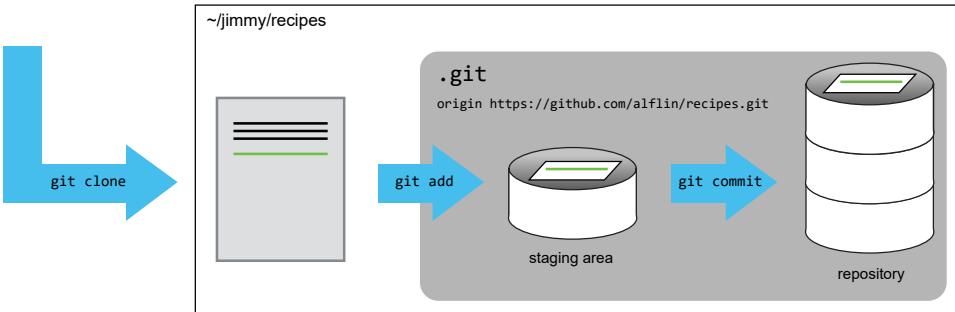
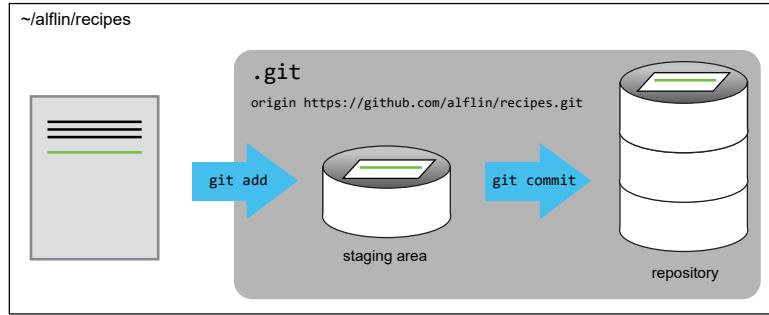
```
git pull
```



Cloning Repositories

- A repository can be cloned from a remote repository with:

```
git clone < URL >
```



Branches

- `git` has a powerful branching system that allows for multiple versions of the repository to be worked on simultaneously.
- The default branch is called `main`.
- A new branch can be created with:

```
git branch < branch-name >
```

- The branch can be switched with:

```
git checkout < branch-name >
```