Introduction to Git

Version control using git

Patrick Diehl
LANL HPC training

LA-UR-YY-NNNNN

Motivation

Version control of source code, papers, grants, or other documents is very important to track changes. Note that most version control systems do not work well with MS Office. However, it works well with LaTeXand Overleaf has some integration for git.

Where do we use use git at LANL

- Overleaf (`https://sharelatex.lanl.gov/saml/login`)
- Internal GitLab (`https://gitlab.lanl.gov`)
- Public GitHub (`https://github.com/lanl`)
- Many other places...
- Many open source projects are using git, e.g. Spack, Kokkos, Linux kernel 😄

git

--distributed-even-if-your-workflow-isnt

Git is a

- free and open source
- distributed version control system
- designed to handle everything
- from small to very large projects
- with speed and efficiency.

Note that you can use git locally without the need of a server.

Outline

Terminology

Using git to version control projects

Using git to collaborate with others

Neat git features 😆

Advanced git features 🤪

Terminology

Terminology I

Commits

A commit in a git repository records a snapshot of all the (tracked) files in your directory. It's like a giant copy and paste, but even better!

Git wants to keep commits as lightweight as possible though, so it doesn't just blindly copy the entire directory every time you commit. It can (when possible) compress a commit as a set of changes, or a "delta", from one version of the repository to the next.

History

Git also maintains a history of which commits were made when. That's why most commits have ancestor commits above them – we designate this with arrows in our visualization. Maintaining history is great for everyone working on the project!

Terminology II

Branch

Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.

Merge

Merging means that the encapsulated changes from a branch are applied (merged) to another branch. if these changes of the branch to be merged conflict with the state of the other branch, we get merge conflicts and we have to resolve these conflicts.

Merge conflict

Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it. In these cases, Git cannot automatically determine what is correct.

Using git to version control projects

Initialize your first local git repository

We initialize a local git repository in the folder my_project using `git init`

```
mkdir my_project
cd my_project
# Initialize the local git repository
git init
Initialized empty Git repository in /Users/diehlpk/my_project/.git/
```

We check the status if the repo using `git status`

```
git status
On branch main

No commits yet

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

We see there were no commits yet and that we are on the master branch.

Adding files and committing them to the history

We add a file to be tracked by git using `git add`

```
touch my_file.txt
git add my_file.txt
# Check the status of the git repo repository
git status
On branch main

No commits yet

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

We commit the changes using `git commit`

```
git commit -m "Adding my fist file"
[main (root-commit) dbad690] Adding my fist file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 my_file.txt
```

Adding changes to the previously added file

We use `git commit` to add the changes to the git history

```
echo "Patrick made a typo" >> my_file.txt
git commit -m "Add new content" my_file.txt
[main fb2555c] Add new content
 1 file changed, 1 insertion(+)
```

We show the history using `git log`

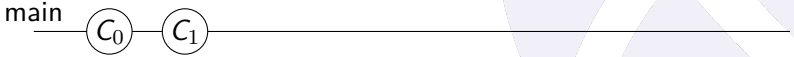
```
git log
commit fb2555c430314cb9d8ea3d3c8f867fdb7faa488e (HEAD -> main)
Author: Patrick Diehl <diehlpk@lanl.gov>
Date:   Fri Aug 23 11:33:01 2024 -0600
    Add new content

commit dbad6907a2a3348b452149342f70789cbbfbe060
Author: Patrick Diehl <diehlpk@lanl.gov>
Date:   Fri Aug 23 11:29:10 2024 -0600
    Adding my fist file
```

Show differences between the two commits

We use `git diff` to show the changes between the two commits:

```
git diff fb2555c430314cb9d8ea3d3c8f867fdb7faa488e dbad6907a2a3348b452149342f078
diff --git a/my_file.txt b/my_file.txt
index fcb8ed9..e69de29 100644
--- a/my_file.txt
+++ b/my_file.txt
@@ -1 +0,0 @@
-Patrick made a typo
```

Let us visualize the git history...



Make another commit

We use `git diff` to show the changes between the two commits:

```
echo "Patrick made another change!" >> my_file.txt
~/git-tut{main} $ git commit -m "Add a new reference" my_file.txt
[main b85b51d] Add a new reference
 1 file changed, 1 insertion(+)
```

Let us visualize the git history...


```
cat my_file.txt
Patrick made a typo
Patrick made another change!
```

Now, we have three commits on the main branch. Ops, I just realized that I introduced a typo in commit C_1 and I want to remove this commit from the history.

Remove last commit from history

We use `git rebase` with (`HEAD~`) to remove the last commit

```
git reset --hard HEAD~
```

Let us visualize the git history...


```
cat my_file.txt
Patrick made a typo
```

Note the last line added by the last commit was removed from `my_file.txt`.

Using a branch to work on new content

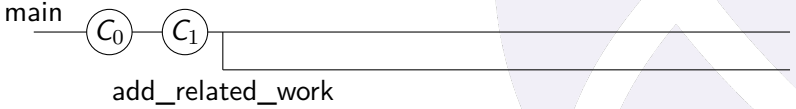
We use `git branch` to show all current branches

```
git branch
* main
```

We use `git branch add_related_work` to show all current branches

```
git branch add_related_work
git branch
  add_related_work
* main
```

Let us visualize the git history...



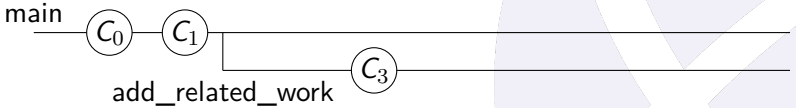
We use `git checkout` to switch to the new branch

```
git checkout add_related_work
```

Updating a branch

```
echo "Related work added" >> my_file.txt
git commit -m "Add related work"
[main 6ee4289] Add related work
 1 file changed, 1 insertion(+)
```

Let us visualize the git history...



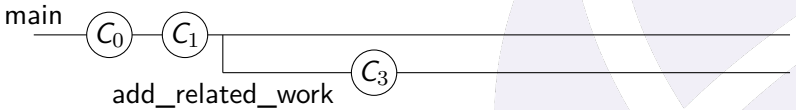
Tthe file `my_file.txt` has the following state (content) in the branch

```
cat my_file.txt
Patrick made a typo
Related work added
```

Switching branches

```
git checkout main
Switched to branch 'main'
```

Let us visualize the git history...



The file `my_file.txt` has the following state (content) in the main branch

```
cat my_file.txt
Patrick made a typo
```

So all changes in the branch `add_related_work` are local to the branch!

Merging branches

We can use (git diff) with branches to showcase the differences

```
git diff main add_related_work
diff --git a/my_file.txt b/my_file.txt
index 3cb3c28..e69de29 100644
--- a/my_file.txt
+++ b/my_file.txt
@@ -1,1 +0,0 @@
-Related work added
```

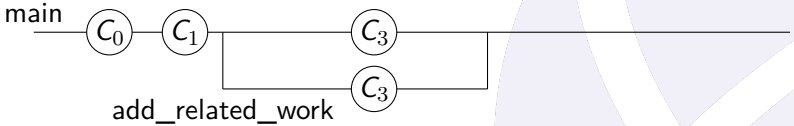
We can use (git merge) to merge the changes of the branch `add_related_work` to the main branch

```
git checkout main
git merge add_related_work
Updating 71e2378..68f05f5
Fast-forward
 my_file.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Merging branches

```
cat my_file.txt
Patrick made a typo
Related work added
```

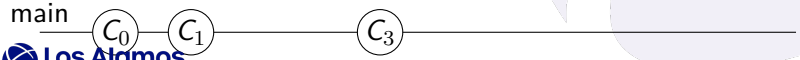
Let us visualize the git history...



We use `git branch -d` to delete a branch

```
git branch -d add_related_work
Deleted branch add_related_work (was 68f05f5).
```

Let us visualize the git history...



Reverting uncommitted changes changes to files

We changed the file `my_file.txt` and have not yet committed the change

```
cat my_file.txt
Patrick made a typo
Related work added
Make a new edit
```

We use `git stash` to revert to the last commit

```
git stash
Saved working directory and index state WIP on main: 68f05f5 Add related wor
```

And now the added line was removed

```
cat my_file.txt
Patrick made a typo
Related work added
```

Untracking a file

We use `git rm` to untrack a file from the history

```
git rm my_file.txt
rm 'my_file.txt'
```

We need to use `git commit -a` to remove the file from the history

```
Delete file
   # Please enter the commit message for your changes. Lines starting
   # with '#' will be ignored, and an empty message aborts the commit.
   #
   # On branch main
   # Your branch is up to date with 'origin/main'.
   #
   # Changes to be committed:
   #       modified:   git.tex
   #       deleted:    my_file.txt
```

We see in the git commit message that the file will be deleted with this commit.

Summary of git commands

- `git init` – Initialize the local git repository
- `git status` – Show the current branch and commits
- `git add filename` – Add the file with the name filename to git's history
- `git commit -m "message" filename` – Add the changes of the file with the the name filename to git's history
- `git commit -a` – Add all changes of all tracked files to git's history
- `git log` – Show the current commits in git's history
- `git diff` – Show the differences between files or branches
- `git revert HEAD` – Remove the last commit
- `git branch` – List all branches
- `git branch name` – Generate a new branch from the current state
- `git checkout name` – Checkout to the branch name
- `git branch -d name` – Delete the branch name

Exercise

Please do the following

- Initialize a git repository on your personal device or on the cluster
- Add a text file to the repo
- Commit the empty file to the git history
- Add content to the file and add commit the content to the git history
- Generate a new branch and add commits to the branch
- Look at the differences between the main branch and your branch
- Merge your branch in the main branch

Using git to collaborate with others

Using a git server to collaborate with others

- Previously, we used git to version control local files
  - Files are only local and there is no backup
  - We can not collaborate with others on the same files
- There are commercial services, like GitHub or BitBucket, or LANL hosted services, like gitlab.lanl.gov, providing a git server and a website for project management, like issue tracker, reviews, and many more features.
  - Now, we have the global state of the tracked files on the server
  - As before we have the local stage of the tracked files
- Now, if we collaborate with other users, every user has its own stage on his local device and all users synchronize with the git server,

Retrieving the state of the git server

We use `git clone` to get the current state of the server

```
git clone git@gitlab.lanl.gov:ic-user-training/introduction-git.git
Cloning into 'introduction-git'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 21 (delta 4), reused 0 (delta 0), pack-reused 3 (from 1)
Receiving objects: 100% (21/21), 322.68 KiB | 8.07 MiB/s, done.
Resolving deltas: 100% (4/4), done.
```

Note that we pulled the repo with the LaTeXsources for the slides for this course.

```
ls
introduction-git
```

Now there is a folder with the same name of the repository with the current state on the git server.

The main branch

```
git branch
* main
```

The main branch on the git server is the current state, we cloned.

We use `git remote show origin` to verify the origin git server

```
 git remote show origin
* remote origin
  Fetch URL: git@gitlab.lanl.gov:ic-user-training/introduction-git.git
  Push  URL: git@gitlab.lanl.gov:ic-user-training/introduction-git.git
  HEAD branch: main
```

We fetch all global states and push our local states to the same server. The HEAD branch is the main branch and all contributors should merge their changes into this branch.

Pushing a local branch to the git server

We use the same commands to generate a git branch

```
git checkout -b collaboration
Switched to a new branch 'collaboration'
```

We use `git push` to send potential changes in the branch to the git server

```
git push
fatal: The current branch collaboration has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin collaboration
```

However, git tells us that the branch is only local and not yet on the upstream (git server). Git is very user-friendly and shows in the error message what to do to fix it.

Pushing a local branch to the git server

We use `git push --set-upstream` to add the branch to the git server

```
 git push --set-upstream origin collaboration
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for collaboration, visit:
remote:   https://gitlab.lanl.gov/ic-user-training/introduction-git/-/merge_
remote:
To gitlab.lanl.gov:ic-user-training/introduction-git.git
 * [new branch]      collaboration -> collaboration
branch 'collaboration' set up to track 'origin/collaboration'.
```

Now the branch is on the server with the current local state of your branch. Note that you have to do `git push` to push your new changes to the server.

Let people know who you are

We use `git config --global user.name "value"` to set our user name

```
git config --global user.name "Robert Oppenheimer"
```

We use `git config --global user.name` to show the current value

```
git config --global user.name
  Robert Oppenheimer
```

We use `git config --global user.email "value"` to set our email address

```
git config --global user.email "ro@lanl.gov"
```

For collaborative project having a name and email address in the history is beneficial to see who did the changes and how to reach out to them.

Deleting branches

We use `git branch -d` to delete the local branch

```
git branch -d collaboration
Deleted branch collaboration (was 31a7a1f).
```

We use `git push origin -d` to delete the branch on the git server

```
git push origin -d collaboration
To gitlab.lanl.gov:ic-user-training/introduction-git.git
 - [deleted]         collaboration
```

Pulling changes from the git server

We use `git pull` to synchronize the state on the server with our local state

```
git pull
Enter passphrase for key '/Users/diehlpk/.ssh/id_ed25519':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 272 bytes | 30.00 KiB/s, done.
From gitlab.lanl.gov:ic-user-training/introduction-git
   31a7a1f..e434ab2  main       -> origin/main
Updating 31a7a1f..e434ab2
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Here, we see that the file `README.md` had changes on the git server which were applied to my local state.

Squashing multiple commits into one single commit

We use `git rebase -i HEAD n` to get the `n` last commits

```
<<<<<<< Updated upstream
git rebase -i HEAD~3
=======
git rebase HEAD~3
   pick 84b84f6 Add git pull
  pick 3c741c3 Add rebase
  pick 3580693 Fix typo
>>>>>>> Stashed changes
```

That will open your default editor and listing thee last three commits

```
  pick 84b84f6 Add git pull
  pick 3c741c3 Add rebase
  pick 3580693 Fix typo
```

For the last two commits the `pick` will be edited to `squash` (s)

```
  pick 84b84f6 Add git pull
  s 3c741c3 Add rebase
  s 3580693 Fix typo
```

After exiting the editor, git will open a new editor and let you change the commit

Squashing multiple commits into one single commit

After editing the commit message, you will see

```
Add new content
 Date: Tue Sep 3 13:45:08 2024 -0600
 1 file changed, 64 insertions(+)
Successfully rebased and updated refs/heads/main.
```

Now we run `git log`

```
git log
commit 5ba26dae1613a55ce5260b528f60dc1eccac0d6b (HEAD -> main)
Author: Patrick Diehl <diehlpk@lanl.gov>
Date:   Tue Sep 3 13:45:08 2024 -0600

    Add new content
```

And the previous three commits are removed from the history and the new commit including all three commits is shown. This feature is useful to combine smaller changes into a larger commit to be pushed to the git server.

Deleting a commit

We use `git rebase -i HEAD n` to get the n last commits

```
git rebase -i HEAD~2
```

That will open your default editor and listing thee last two commits

```
  pick ed3e953 Add squash
  pick 169f677 Update README.md
```

For the last commit to be deleted the `pick` will be edited to `delete` (d)

```
  pick ed3e953 Add squash
  d 169f677 Update README.md
```

After exiting the editor, git will show

```
Successfully rebased and updated refs/heads/main.
```

And the changes introduced by this commit will be removed from the current state.

Resolving a merge conflict

We use `git merge main` to merge the state of the main branch to my local branch

```
git merge main
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Git is very user-friendly and will let us know that the file `README.md` has a merge conflict. Let us look into the file

```
* Using git to collaborate with others
* Advanced git features
 <<<<<<< HEAD
 Let us introduce an accidental conflict
 =======
 >>>>>>> main
```

We see that in the main branch the line was deleted but us still here in our branch. So we need to edit the file accordingly and decide which changes we want to have.

Resolving a merge conflict

After editing the file and resolving the conflict, we run

```
git commit -a
```

Git will open your default editor and shows

```
Merge branch 'main' into collaboration

# Conflicts:
#       README.md
#
```

After you exit the editor, you will see

```
[collaboration 343fbe5] Merge branch 'main' into collaboration
```

Now the conflict is resolved the the main branch was merged into your branch

Neat git features 😆

Nice visualization of the history

```
git log --graph --decorate --oneline
* fe6fef8 (HEAD -> main) Resotre
* 7d33da4 (origin/main, origin/HEAD) Add git rm
* dbbc232 Delete file
* 3a3e20a Add file
* da172e4 (collaboration) Add git resolve
*   343fbe5 Merge branch 'main' into collaboration
|\
| * 500f5f4 Help with conflict
* | 8e31113 Another commit
|/
* 234bb1e add conflict
* ed3e953 Add squash
* 5ba26da Add new content
* e434ab2 Update README.md
```

Nice visualization of the history

```
git log --graph --decorate --oneline
* fe6fef8 (HEAD -> main) Resotre
* 7d33da4 (origin/main, origin/HEAD) Add git rm
* dbbc232 Delete file
* 3a3e20a Add file
* da172e4 (collaboration) Add git resolve
*   343fbe5 Merge branch 'main' into collaboration
|\
| * 500f5f4 Help with conflict
* | 8e31113 Another commit
|/
* 234bb1e add conflict
* ed3e953 Add squash
* 5ba26da Add new content
* e434ab2 Update README.md
```

Format the git log

Get the last commit of the history with the short hash

```
git log -1 --abbrev-commit
commit 18227f1 (HEAD -> main)
Author: Patrick Diehl <diehlpk@lanl.gov>
Date:   Tue Sep 3 16:15:59 2024 -0600
    Add visualization
```

Get the last commit of the history with the short hash and condensed in one line

```
git log -1 --abbrev-commit --oneline
18227f1 (HEAD -> main) Add visualization
```

Format the complete log

```
git log  --abbrev-commit --oneline
18227f1 (HEAD -> main) Add visualization
e0d5bd2 (origin/main, origin/HEAD) Add vis
fe6fef8 Restore
7d33da4 Add git rm
dbbc232 Delete file
```

Change the last commit message

```
git log
commit e0d5bd262924d4620a5fbac8d63550a08551466d (HEAD -> main, origin/main,
Author: Patrick Diehl <diehlpk@lanl.gov>
Date:   Tue Sep 3 16:00:15 2024 -0600
    Add vis
```

We can use `git commit --amend` to change the last commit message

```
git commit --amend -m "Add visualization"
[main 0572ae8] Add visualization
 Date: Tue Sep 3 16:00:15 2024 -0600
 1 file changed, 24 insertions(+)
```

```
git log
commit e0d5bd262924d4620a5fbac8d63550a08551466d (HEAD -> main, origin/main,
Author: Patrick Diehl <diehlpk@lanl.gov>
Date:   Tue Sep 3 16:00:15 2024 -0600
    Add visualization
```

Create an archive

Generate a tar archive from the current HEAD ç Generate a zip archive from the branch collaboration

```
git archive --format=zip HEAD > project.zip
```

Generate a zip archive from the main branch of remote repository

```
git archive --remote=https://example.com/myrepo.git main | gzip > spack.zip
```

Configure the files git is tracking

Sometimes we do not want to track certain files with git. For example for code projects, we want to track the source files, scripts, and build system files. However, we do not want to track compiled files, like executables or libraries.

To specify these files, we can add a file `.gitignore` in the main folder of the git repository.

Examples for most common programming languages are available here:
`https://github.com/github/gitignore/tree/main`

Advanced git features 🤪

Tagging states for releasing versions

Generate the release v0.1 from the current state with a release message

```
git tag v0.1 -m "Initial release"
```

Generate the release v0.1 from a commit hash with a release message

```
git tag -a v1.2 9fceb02 -m "Some very cool new feature"
```

Push one tag to the git server

```
git push origin v1.5
```

Push all new tags to the git server

```
git push origin --tag
```

Delete the tag v0.1

```
git tag -d v0.1
```

Generating and apply patches

Sometimes we want to extract the changes of a single commit and apply these to a different state. We use `git diff` to get the changes and pipe them to the file `archive.patch`

```
1 git diff ad3a6a3 > archive.patch
```

Now, we use `git apply` to add these changes from the commit to another state

```
1 git apply archive.patch
```

Get the changes between branches and pipe them to a file

```
1 git diff main collaboration > bigger.patch
```

This feature is for example needed to extract small patches fir bug fixes after a release and apply these before the next release. The Fedora package systems uses patches to fix smaller issues with releases.

Removing one commit from the current state but keep the commit in the history

We added a bad commit (69f85d8) to our project

```
git log --oneline
9c8778f (HEAD -> main) Good update
69f85d8 Update with issue
96942b5 Add Readme
```

We can use `git revert` to remove the changes of the commit

```
git revert 69f85d
```

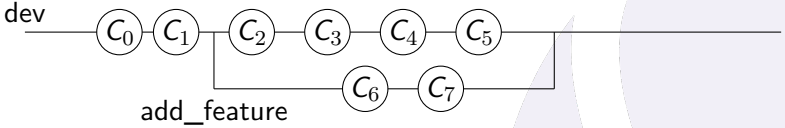
The commit is still in the history but the changes are removed from b25124f

```
 git log --oneline
b25124f (HEAD -> main) Revert "Update with issue"
9c8778f Good update
69f85d8 Update with issue
96942b5 Add Readme
```

This feature is nice to keep the bad commit documented but have it removed.

Cherry pick one commit

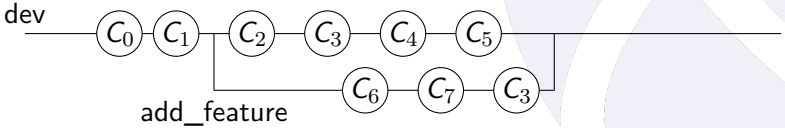
Let us visualize the git history...



Sometimes, we want to pick one commit from a branch and add it to another branch

```
1 git checkout add_feature
2 git cherry-pick dev~2
```

Let us visualize the git history...



Find a bug introduced somewhere in the commit history

We know that the code worked with commit 491d2b5 and we found a bug in the current HEAD. We could have 100 commits between these commits. We could checkout each of them by hand to find the red herring. To make that easier we use

```
git bisect start
git bisect bad % HEAD is the default or use a hash of a commit
git bisect good 491d2b5
```

This checks out the commit in the middle of the good and bad commit

```
make
ctest
```

We can test the current commit and see if the code still works. We can use

```
git bisect good # Jumps to the middle of current and good
git bisect bad   # Jumps to the middle of current and bad
```

We stop the current bisect using

```
git bisect reset # Stop
```

Thanks for your attention and I am happy to answer remaining questions.

You can contact me at diehlpk@lanl.gov or @patrickdiehl on Mattermost if needed.

Thanks to Richard Berger for his feedback on the course.

If you missed topics or would have heard more about one topic, please reach out to me. I will try to adapt the course for the next iteration.