GIT

Getting Started

Task	Command
Start a new repo	git init
Clone an existing repo	git clone <url> git clone http://192.168.178.96:3000/<Repository Name>

Configure Git

Task	Command
Set a config option	git config user.name 'Your Name'
Set a config option	git config user.email 'Your email'
Set option system wide in /etc/gitconfig	git config –system …
Set option globally in ~/.gitconfig	git config –global …
Set option locally in the repository .git/config file	git config –local …
Add an alias	git config alias.st status
See all possible config options	man git-config

Prepare to Commit

Task	Command
Add untracked file or unstaged changes	git add <file>
Add all untracked files and unstaged changes	git add .
Choose which parts of a file to stage	git add -p
Move file	git mv <old> <new>
Delete file	git rm <file>
Tell Git to forget about a file without deleting it	git rm –cached <file>
Unstage one file	git reset <file>
Unstage everything	git reset
Check what you added	git status
Check what you added, short format	git status -s

Make Commits

Task	Command
Make a commit (and open text editor to write message)	git commit
Make a commit	git commit -m 'message'
Commit all unstaged changes	git commit -am 'message'

Branches

Task	Command
List branches	git branch
List branches by most recently committed to	git branch –sort=-committerdate

Task	Command
Switch branches	git switch <name> OR git checkout <name>
Create a branch	git branch <name> OR git switch -c <name> OR git checkout -b <name>
Delete a branch	git branch -d <name>
Force delete a branch	git branch -D <name>

Diff Staged/Unstaged Changes

Task	Command
Diff all staged and unstaged changes	git diff HEAD
Diff just staged changes	git diff –staged
Diff just unstaged changes	git diff

Diff Commits

Task	Command
Show diff between a commit and its parent	git show <commit>
Diff two commits	git diff <commit> <commit>
Diff one file since a commit	git diff <commit> <file>
Show a summary of a diff	git diff <commit> –stat git show <commit> –stat

Discard Your Changes

Task	Command
Delete unstaged changes to one file	git restore <file> OR git checkout <file>
Delete all staged and unstaged changes to one file	git restore –staged –worktree <file> OR git checkout HEAD <file>
Delete all staged and unstaged changes	git reset –hard
Delete untracked files	git clean
'Stash' all staged and unstaged changes	git stash

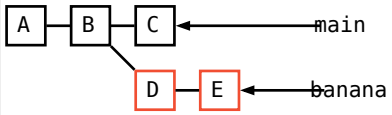
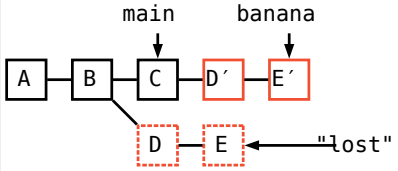
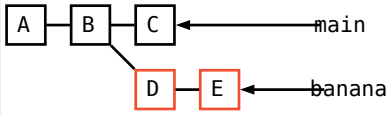
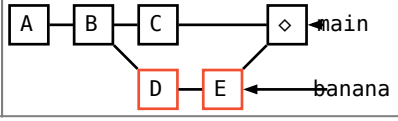
Edit History

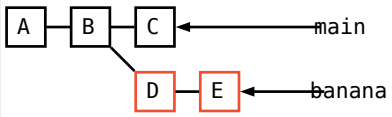
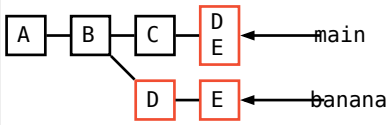
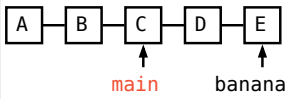
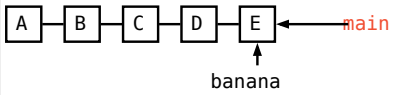
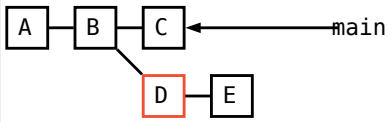
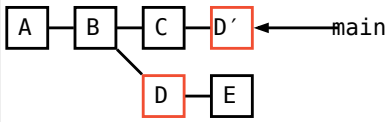
Task	Command
“Undo” the most recent commit (keep your working directory the same)	git reset HEAD^
Squash the last 5 commits into one	git rebase -i HEAD~6 Then change “pick” to “fixup” for any commit you want to combine with the previous one
Undo a failed rebase	git reflog BRANCHNAME Then manually find the right commit ID in the reflog, then run: git reset –hard <commit>
Change a commit message (or add a file you forgot)	git commit –amend

Code Archaeology

Task	Command
Show the files under control	git ls-files
Look at a branch's history	git log main git log –all –graph git log –graph <main> git log –oneline
Show every commit that modified a file	git log <file>
Show every commit that modified a file, including before it was renamed	git log –follow <file>
Find every commit that added or removed some text	git log -G banana
Show who last changed each line of a file	git blame <file>

Combine Diverged Branches

Task	Command
Abort a merge	git merge –abort
Combine with Rebase	git switch banana git rebase main Before  After 
Combine with Merge	git switch main git merge banana Before  After 

Task	Command
Combine with Square Merge	git switch main git merge –squash banana git commit Before  After 
Bring a branch up to date with another branch (aka “fast-forward merge”)	git switch main git merge banana git commit Before  After 
Copy one commit onto the current branch	git cherry-pick <commit> Before  After 

Restore an Old File

Task	Command
Get the version of a file from another commit	git checkout <commit> <file> OR git restore <file> –source <commit>

Remote Repositories

Task	Command
Add a Remote	git remote add <name> <url>
Push the main branch to the remote origin	git push origin main
Push the current branch to its remote “tracking branch”	git push
Push a branch that you've never pushed before	git push -u origin <name>
Force push	git push –force-with-lease

Task	Command
Push tags	git push –tags
Pull changes (but don't change any of your local branches)	git fetch origin main
Pull changes and then rebase your current branch	git pull –rebase
Pull changes and then merge them into your current branch	git pull origin main OR git pull

Submodules

Adding a Submodule

Navigate to the root directory of ‘main-project’ and run the following command:

```
git submodule add <mysubmodule-url>
```

This command adds ‘mysubmodule’ as a submodule to ‘main-project’. It clones ‘mysubmodule’ into a subdirectory of ‘main-project’ and creates a special file called **.gitmodules**. This file tracks the mapping between the project URL and the local subdirectory you've pulled it into.

After adding the submodule, you need to commit this change to your repository:

```
git commit -m "Added util-methods submodule"
```

Now, you can import and use function from 'mysubmodule' in your 'main-project' just like you would with any local module.

Updating a Submodule

Navigate to the ‘mysubmodule’ directory within ‘main-project’ and pull the latest changes:

```
cd mysubmodule
git pull origin master
```

After pulling the latest changes, navigate back to the root directory of ‘main-project’ and commit this update:

```
cd ..
git commit -m "Updated mysubmodule submodule"
```

Alternatively, you can use the git submodule update –remote command from the root directory of 'main-project' to update all submodules to their latest commiton their master branches. Remember to commit after running this command as well.

Submodule Caveats

There are some caveats and things to keep in mind when using Git submodules:

- Updating: Submodules do not automatically stay in sync with the remote repository. You need to go into each submodule and pull the changes manually.
- Committing: If you make changes inside a submodule, you need to commit those changes within the submodule first, then go to the main repository and commit the change to the submodule. This is because the main repository tracks the submodule's commit.
- Cloning: When you clone a repository, the submodules will not be cloned along with it. You need to use git submodule update –init –recursive or git clone –recursive <repository> to clone the repository and its submodules.
- Pulling: Similarly, when you pull your main repository, it will not pull the new commits of populated submodules. Use the –recurse-submodules option with the git pull to update all the submodules as well.

Important Files

Item	Location
Local git config	.git/config
Global git config	~/.gitconfig
List of files to ignore	.gitignore

Merge Conflicts

Common merge error:

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

Run command:

```
git log --merge --oneline
----------------
1d91b52 (bugfix1) 2e bugfix
91d17b9 (HEAD -> master) New featurein Master
578b59d bugfix in bugfix branch
```

```
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.c
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  test_BACKUP_29859.c
        test_BASE_29859.c
        test_LOCAL_29859.c
```

```
        test_REMOTE_29859.c
```

Edit the test.c file to resolve the conflict:

```
nano test.c
-----------

first version test.c file
<<<<<<< HEAD
New Feature added in Master branch
=======
Bugfix1 in bugfix branch

Bugfix2 in bugfix branch
>>>>>>> bugfix1
```

Links

- Cheat Sheet