Git Basics

What is Git?

A decentralized version control system created by Linus Torvalds.

What is a Version Control System

A system that internally keeps track of the many versions of files you may create to manage changes across time.

Why is it used?

It is one of the most complex yet the most feature rich version control systems.

What problem does it solve?

Git keeps track of historical changes you make to files across time so you can, relatively, easily revert back to a prior version, collaborate with a team and view modifications.

Decentralized approach to version control

> Other version control systems have relied on having one master copy of a file exist on a server whereas git stores the entire repository history on every copy on every computer which has access to it.

What is a Repository

> A environment that is managed by git that houses all changes to a particular project across time.

General Resources

> Git: Homepage
>
> Git: The Simple Guide

Install

Linux / Mac

> 👍 You already have some version of it it installed!

Windows

You must download Git Bash

Upgrading to Latest

Atlassian: Installing and Upgrading Git

First Time Setup

The firs time you setup git, you will need to configure your git profile by changing your user name and user email via the following commands.

```
# replace "John Doe" with your name
# replace johndoe@example.com with your email
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Basic Commands

```
# fires up help text
git help

# shows help text only for specific command
git help | grep command


# initializes a new git repository
git init


# stages the commit
git add file_name
```

```
# commits changes to your local repo
# will fire up default terminal text editor for commenting (default is usually vim)
git commit

# commit changes to your local repo along with a commit message
git commit -m "Commit message"


# checks the status of your repository
# displays information on untracked, unstaged, unmerged files
git status

# displays commits in reverse chronological order
git log

Git log --oneline

# creates a new branch based on the branch_name
git branch branch_name

# switches to a branch_name
git checkout branch_name

# deletes branch_name
git branch -d branch_name

# deletes branch_name and ignores unmerged warnings
git branch -D branch_name

# clones git repository
# used alongside Github, Bitbucket, etc
git clone git_url
```

Undo Commands

```
# checks out a specific commit
git checkout d5c89348d77a

# creates a new branch based on this commit
# the HEAD pointer is pointing to the new branch
git checkout -b new_branch_without_crazy_commit

# inverses changes from last commit, creates a new commit with these inversed changes
git revert

# inverses changes from last commit, creates a new commit
git reset
```

Resource

Atlassian: Undoing Commits & Changes

Remote Commands

```
# list remote repos along with urls
git remote -v

# adds remote repo to workspace
git remote add remote_name remote_url
```

```
# pushes local code to remote branch
git push remote_name branch_name

# fetches updated changes from remote
git fetch remote_name

# merge code from branch_name into current branch
git merge branch_name

# pulls local code to remote branch
git pull remote_name branch_name

# pulls local code to remote branch
# same as prior command in two steps
git fetch remote_name
git merge remote_name/branch_name
```

Workflow

1. Initialize repository
2. Create file
3. Stage changes to be committed
4. Commit file to tree
5. Make changes to files
6. Commit changes
7. Profit

Tips

- Commit often
- Atomic Commits: Keep your commits small

Related Technology

- CVS
- SVN

Exercise

Setup

Create a new directory called git_practice

Instructions

Initial Commit

1. Initialize a new git repository
2. Create a file called index.html with an html `<head>` and `<body>`
3. Add the following text into the html body:

```
<p>Hello World!</p>
```

1. Check the status of the git repository
2. Stage the file
3. Check the status of the git repository
4. Commit the file with a message of "First commit"

Second Commit

1. Create a new file in the same directory named main.css
2. Add the following text into the css body: p { color: blue; }
3. Link the main.css in the html file
4. Open the html file in the browser to make sure it linked css file
5. Check the status of the git repository Stage the file
6. Check the status of the git repository Commit the file with a message of "Added main.css file"

Git Resources

1. Getting Started with Git
2. Git

Git Cheatsheet

fournova

GIT CHEAT SHEET

presented by **TOWER** › Version control with Git - made easy

CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit

...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```