

# এক পলকে গিট ৩ গিটহাব

জুনায়েদ আহমেদ



জুনায়েদ

# এক পলকে গিট

## ও গিটিহাব

জুনায়েদ আহমেদ

প্রথম এডিশন

উইব জুনায়েদ

WITH.ZONAYED.ME

# কপিরাইট সম্পর্কিত কথা

প্রথমেই আপনাকে অনেক অনেক ধন্যবাদ বইটি সংগ্রহ করার জন্য। যদিও বইটি সম্পূর্ণ বিনামূল্যে উইদ জুনায়েদ প্ল্যাটফর্মে এভেইলেবল রয়েছে, কিন্তু এটা ব্যবহারের ক্ষেত্রে কিছু সাবধানতা অবলম্বন করার জন্য অনুরোধ থাকলো।

প্রথমতঃ এই বইয়ের সব কন্টেন্ট উইদ জুনায়েদ প্ল্যাটফর্মের কাছে সর্বস্বত্ত্ব সংরক্ষিত। তাই এই বইয়ের কন্টেন্ট দয়া করে কেউ কপি করে অন্যকোথাও ব্যবহার করবেন না। বইটি সবসময়ের জন্য বিনামূল্যে উইদ জুনায়েদ প্ল্যাটফর্মে পাওয়া যাবে। তবে কেউ যদি ব্যাক্তিগত উদ্দেশ্যে, সহজে পড়ার জন্য বইটি প্রিন্ট করতে চান তাহলে সেটা করতে পারবেন। তবে এটা শুধুমাত্র ব্যাক্তিগত উদ্দেশ্যেই হতে হবে এবং অবশ্যই সেইসাথে শেখার উদ্দেশ্য থাকতে হবে।

তবে বইটি কোনোভাবেই কমার্শিয়ালি প্রিন্ট করে বিক্রি অথবা সম্পূর্ণ ফ্রিতেও ডিস্ট্রিবিউট করা যাবে না, ধন্যবাদ।

# লেখকের কিছু কথা

একজন ইভিপেন্ট, প্রোগ্রামিং সম্পর্কিত ইবুক পাবলিশার হিসেবে  
আমি আপনাদের সবার সাথে এই বইটি শেয়ার করতে পেরে খুবই  
আনন্দিত।

আপনারা জেনে অত্যন্ত খুশি হবেন যে আমি আমার প্ল্যাটফর্ম থেকে  
ভবিষ্যতে এরকম টুকটাক বিষয়ের উপর আরো বই আপনাদেরকে  
উপহার দেওয়ার চেষ্টা করব। আমার এই বইগুলো প্রকাশের মাধ্যমে  
আরো বিশাল রেঞ্জের প্রোগ্রামিং ও সফটওয়্যার ইঞ্জিনিয়ারিং রিলেটেড  
বিভিন্ন বিষয়সমূহ আপনাদের সামনে সহজ, সরল এবং সাবলীল ভাষায়  
তুলে ধরার চেষ্টা করব।

তবে একজন ইভিপেন্ট পাবলিশার হিসেবে আমার এই কাজ এবং  
যাবতীয় বিষয়সমূহ কতদুর আগাবে সেটা সম্পূর্ণরূপে আপনাদের  
সাপোর্টের উপরই নির্ভর করছে। আমি যদি আপনাদের সবার কাছ  
থেকে এই বিষয়ে ভালো সাপোর্ট পাই, তাহলে সেটা আমাকে ভবিষ্যতে

আরো অনেক গুরুত্বপূর্ণ বিষয়ের উপর ইবুক পাবলিশ করতে  
অনুপ্রেরণা যোগাবে।

সবশেষে আপনাকে ধন্যবাদ কষ্ট করে ইবুকটি সংগ্রহ করার জন্য,  
যেকোনো অভিযোগ, পরামর্শ বা সাজেশন থাকলে তা অবশ্যই  
আমাকে জানানোর জন্য অনুরোধ থাকলো।

ধন্যবাদান্তে,

উইদ জুনায়েদ (with.zonayed.me)

এই বইটি উৎসর্গ করা হলো আমার সহধর্মিনী

## জাকিয়া আখতার

কে...

# সূচিপত্র

## শুরু

০০	কিছু কথা.....	১১
০১	গিট কি?.....	১৩
০২	গিটহাব কি?.....	১৮

## গিট

০০	গিট সেটআপ.....	২২
০১	গিট কনফিগার.....	২৪
০২	গিট রিপোজিটরি সেটআপ.....	২৬
০৩	স্ট্যাটাস চেক করা.....	৩০
০৪	স্টেজিং এরিয়াতে নেওয়া.....	৩২
০৫	ফাইনাল কমিট করা.....	৩৫
০৬	ফাইল মডিফাই করে আবার কমিট.....	৩৭

০৭	পুনরায় মডিফাই করে কমিট.....	৪১
০৮	কমিট লগ চেক.....	৪৮
০৯	পূর্বের ভার্শনে যাওয়া.....	৪৯
১০	ব্রাঞ্চ তৈরি.....	৫০
১১	ব্রাঞ্চ এ চেকআউট.....	৫২
১২	নতুন ব্রাঞ্চে মডিফিকেশন.....	৫৪
১৩	ব্রাঞ্চ মেইনে মার্জ.....	৫৯
১৪	কমিটের সাথে কমিটের পার্থক্য.....	৬১

### **গিটহাব**

০০	গিটহাবের সাথে লিঙ্ক.....	৬৫
০১	গিটহাবে পুশ.....	৬৯
০২	SSH কী সেটআপ.....	৭২
০৩	গিটহাব থেকে পুল.....	৮১
০৪	নিজের প্রোজেক্টে পুল রিকোয়েস্ট.....	৮৩
০৫	গিটহাব থেকে প্রোজেক্ট ক্লোন.....	৮৮
০৬	অন্য প্রোজেক্টে পুল রিকোয়েস্ট.....	৯২

### **প্রোজেক্টে কন্ট্রিবিউট**

০০	প্রোজেক্ট খোঁজা.....	১০০
০১	প্রোজেক্ট ফর্ক.....	১০৫
০২	কন্ট্রিবিউট.....	১০৭
০৩	এখনো শেষ হয়নি.....	১১৭
০৪	<b>সেলিব্রেট</b> 	১১৮

### এক্সপ্লোর গিট

০০	গিট রিস্টোর.....	১২০
০১	গিট স্ট্যাশ.....	১২৫
০২	গিট রিসেট.....	১২৯
০৩	গিট রিভার্ট.....	১৩১
০৪	গিট রিবেস.....	১৩৩
০৫	গিট স্কোয়াশিং.....	১৩৭

### অন্যান্য

০০	গিটহাব ব্যবহার করব না.....	১৪০
০১	গিটহাবে SSH ব্যবহার না করা.....	১৪১
০২	স্টুডেন্টদের জন্য গিটহাব অফার.....	১৪৮
০৩	আরো কিছু.....	১৫০

শুরু

# କିଛୁ କଥା

ଆପଣି ସଦି ଆମାକେ ଜିଜ୍ଞାସା କରେନ ଡେଭେଲପମେନ୍ଟ ଏର ଜଗତେ ସବଚେଯେ ଇୱିଜଫୁଲ ଟୁଲ କୋନଟା, ତାଇଲେ ଆମି ଚୋଖ ବନ୍ଧ କରେ ବଲବୋ ଗିଟ । ଆମାର ସାଥେ ଆରୋ ଅନେକେଇ ହୟତୋ ଏକମତ ହବେନ । ତବେ ଗିଟ ଆସଲେ କତଟା ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଆର କାଜେର ତା ବଲାର ଅପେକ୍ଷାଇ ରାଖେ ନା । ଆମି ଏଇ ଅଧ୍ୟାୟେ ବ୍ୟାସିକଲି ଗିଟ ନିଯେଇ ଆଲୋଚନା କରବ । ଆର ଗିଟହାବ ନିଯେଓ ଏକଟୁ ଆଲୋଚନା ଥାକବେ । ଗିଟ ଆର ଗିଟହାବ ନିଯେ ଯାଦେର କନଫିଡ୍ରୁଶନ ଆଛେ, ଏଇ ଦୁଇଟା ଆସଲେ ଭିନ୍ନ ଦୁଇଟା ଜିନିସ । ଏଖାନେ ମେଇନ କାଜ ଗିଟ ଏର ।

ଗିଟ ଶିଖାର ସମୟ ସବାର ମଧ୍ୟେ କମନ ଯେ ପ୍ରଶ୍ନଟା ପ୍ରଥମେଇ ମାଥାଯ ଆସେ, ଗିଟ କେନ ବ୍ୟବହାର କରବ? କେନ ବାରବାର ଆମାକେ କମାନ୍ତଳାଇନେ କମାନ୍ତ ଦିତେ ହବେ । କେନ ଆମାକେ ଏଇ ଏକ୍ସଟ୍ରା ଆରେକଟା ଜିନିସ ଢୁକାତେ ହବେ

আমার প্রোজেক্টে। এটা কি আসলে টাইম ওয়েস্ট না? প্রথমে সবার মাথায় এটাই আসে, কারন এটার গুরুত্ব আর কাজ সম্পর্কে ধারনা না থাকলে এটাকে অতিরিক্ত একটা টুল হিসাবেই মনে হবে। আমারও প্রথমে তাই মনে হয়েছিলো। আর তাই আমি এই বইয়ে যাতে সবাই এটার গুরুত্বটা অন্তত বুঝতে পারে সেরকম উদাহরণ দিয়ে লিখার চেষ্টা করব।

আর শুরু করার আগে আরেকটা কথা বলতে চাইঃ

---

আমি এখানে যে অ্যাপ্রোচগুলো নিয়েছি এগুলো ছাড়াও সেইম কাজ অনেকভাবে, অনেকরকম কমান্ড দিয়ে করা যায় গিট এ। তাই অন্যকোথাও অন্যরকম কিছু দেখে কনফিউজ হওয়ার কোনো কারণ নাই।

---

# গিট কি?

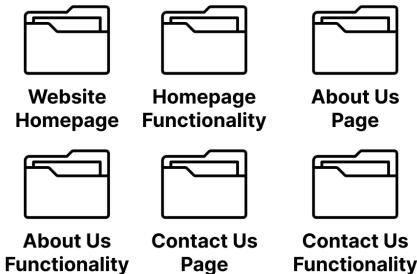
দুই বন্ধু ওয়াফি আর সাহির এর মাথায় খুবই ইন্টারেস্টিং একটা ওয়েবসাইট করার প্ল্যান আসলো। তারমধ্যে ওয়াফি খুব ভালো এইচটিএমএল, সিএসএস লিখতে পারে, একদম পিঙ্কেল পারফেক্ট ওয়েবসাইট এর ইউআই কোড করতে পারে। এবিকে সাহির অ্যাপিআই এবং জাভাস্ক্রিপ্ট দিয়ে বিভিন্ন ফাংশনালিটি খুব ভালো করে করতে পারে।

কিন্তু তারা যে ওয়েবসাইট তৈরি করার প্ল্যান করছে সেখানে দুইটা স্কিলই দরকার। ওয়েবসাইটের ইউআই যেমন ঠিকঠাক হওয়া লাগবে, সেইসাথে অ্যাপিআই এর সাথে কানেকশন, বিভিন্ন ফাংশনালিটিও দরকার। অর্থাৎ তাদের দুইজনকেই একই প্রোজেক্টে কাজ করতে হবে।

প্রথমে ওয়াফিই একদম স্ক্র্যাচ থেকে প্রোজেক্টটা শুরু করলো। প্রয়োজনীয় এইচটিএমএল, সিএসএস লিখে প্রথমেই জাস্ট হোমপেজের কাজ সেরে সবগুলো ফাইল জিপ(.zip) করে সাহিতের কাছে গুগল ড্রাইভের সাহায্যে পাঠালো। সাহিতে এখন সেটা ডাউনলোড করে, আনজিপ করে, ওপেন করে হোমপেজে প্রয়োজনীয় ফাংশানালিটির কাজ করবে।

এরমধ্যে ওয়াফি কিন্তু বসে আছে, কারণ সাহিতে ফাংশানালিটির কাজ শেষ করে তাকে দিলে তারপর সে বাকি পেইজের(অ্যাবাউট আস, টার্মস এন্ড কন্ট্রিন, কন্ট্রাষ্ট আস ইত্যাদি ইত্যাদি) কাজ ধরবে। সাহিতে কাজ শেষ করে আবার আরেকটা ফাইলে প্রোজেক্টটা ওয়াফিকে সেন্ড করলো। এবার ওয়াফি আবার অন্যান্য পেজের ইউআই এর কাজ শুরু করলো, অন্যদিকে সাহিতে ওয়াফির কাজ শেষ হওয়ার জন্য অপেক্ষা করতে থাকলো। এভাবেই তারা তাদের প্রোজেক্টের কাজ আগাছিলো, কিন্তু এখানে কি কয়েকটা সমস্যা খেয়াল করেছেন?

প্রথমতঃ তারা এভাবে ফাইল সেন্ড আর ডাউনলোড করতে করতে তাদের ডাউনলোডস এর ফোন্ডারে এরকম অনেকগুলো ফাইল হয়ে গেছে। সিম্পল রাখার সুবিধার্থে জিপ ফাইলগুলো বাদ দিলাম, নামকরণ ঠিকমতো করলাম আর ধরেন নিলাম একেকটা পেজের মাঝখানে আর কোন চেঞ্জ হয়নি।



### একই প্রোজেক্টের বিভিন্ন ভার্শন

আরেকটা বিষয় খেয়াল করবেন যে ওয়াফি যখন কাজ করছিলো তখন সাহিরকে বসে থাকতে হচ্ছিলো, আবার সেইমভাবে সাহির যখন কাজ করছিলো তখন ওয়াফিকে বসে থাকতে হচ্ছিলো। অর্থাৎ দুইজন কন্টিনিউয়াসলি প্রোজেক্টে একইসাথে কাজ করতে পারছে না। যদি তারা একসাথে কাজ করতোও তাহলে তাদেরকে দুইজনকেই একজনকে আরেকজনের চেঞ্জেসগুলো এনে নিজে নিজে প্রোজেক্টের ভিতর মার্জ করতে হতো।

এসব সমস্যার সাথে আরো হাজারো সমস্যা আছে যেগুলোর কারণে আসলে প্রোজেক্টটা করতে যেমন অনেক কষ্টসাধ্য হয়ে যাবে, সেইসাথে সময় ও পরিশ্রমও অনেক বেশী লাগবে। একে তো আপনি আপনার স্কিল ব্যবহার করে প্রোজেক্ট ইমপ্লিমেন্ট করার পরিশ্রমটা করছেনই, সেইসাথে আপনাকে অন্যান্য ডেভেলপারদের সাথে এটা

সেটা করে প্রোজেক্টটাকে ম্যানেজ করতেই অবস্থা টাইট হয়ে যাবে।  
বরং ছোটবড় যে প্রোজেক্টই করুন না কেন, এটা আসলে সঠিক উপায়  
না। এটার একটা সঠিক উপায় আছে, সেটা হচ্ছে গিট ও গিটহাব  
ব্যবহার করা। তাই প্রথমেই আমরা জানবো গিট আসলে কি?

গিট হলো ভার্শন কন্ট্রোল সিস্টেম। গিটের অনেকগুলো কাজের মধ্যে  
প্রধান এবং প্রাইমারী কাজ হচ্ছে আপনার প্রোজেক্টের প্রত্যেকটা চেঞ্জ  
ট্র্যাক করে রাখা আপনার মন/চাহিদা মতো। আপনাকে বারবার  
প্রোজেক্টের নতুন ভার্শনের জন্য নতুন করে আগের প্রোজেক্ট কপি  
করে আরেকটা নতুন ফোল্ডারে/ডিরেক্টরিতে রাখতে হবে না। আপনি  
জাস্ট কয়েকটা গিট এর কমান্ড দিয়েই চাইলে আপনি আপনার  
প্রোজেক্টের ট্র্যাক করা আগের ভার্শনে চলে যেতে পারবেন।

আবার একদম নতুন ভার্শনেও চলে আসতে পারবেন। এর জন্যে  
আপনার একটা ডিরেক্টরিই থাকবে, প্রত্যেকটা ফাইলেরও একটা কপিই  
থাকবে আপনার প্রোজেক্টের ডিরেক্টরিতে। আপনাকে আপনার  
প্রোজেক্টের ভার্শন চেঞ্জ করার জন্যে কোনো ফাইল হাত দিতে হবে  
না। সব গিট করে দিবে। এখন গিট ব্যবহার করলে খুব সহজেই  
উপরের উদাহরনে একটা ফাইলই থাকতো, কিন্তু চাইলে আবার আমরা  
গিট কমান্ডের সাহায্যে আগের ভার্শনগুলোতেও যেতে পারবো।

এখন এছাড়াও গিট ব্রাঞ্চ সিস্টেম রয়েছে যেটার প্রধান কাজ হলো, আমরা মাঝেমধ্যে আমাদের প্রোজেক্টে নতুন অজ্ঞান কোনো ফিচার অ্যাড করতে চাই। অনেকক্ষেত্রে দেখা যায় আমাদের এই ফিচারটা কেমন লাগবে সেটা সম্পর্কে ধারনা নাই। ভাবি হয়তো একবার ফিচারটা অ্যাড করে নিয়ে দেখলে বলা যাবে আসলে ফিচারটা প্রোজেক্টের সাথে যায় কি যায় না। সেক্ষেত্রে গিট ছাড়া হয়তো আমরা আমাদের মেইন প্রোজেক্টেই সেটা অ্যাড করতাম। তারপর টেস্ট করতাম কেমন হয়েছে সেটা দেখার জন্যে। তারপর ভালো না লাগলে আবার সব জায়গায় গিয়ে গিয়ে ম্যানুয়ালী নতুন লেখা কোডগুলো মুছে ফেলতে হতো, ফাইল ডিলেট করতো হতো। আর ভালো লাগলে ব্যাস এভাবেই রেখে দিতে হতো।

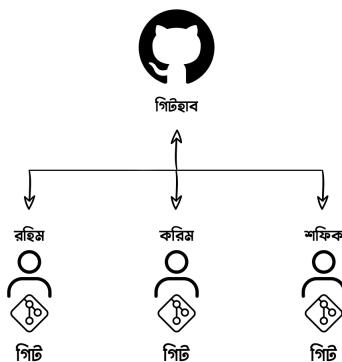
কিন্তু গিট এ ব্রাঞ্চিং এর মাধ্যমে আমরা আমাদের প্রোজেক্টের আরেকটা ব্রাঞ্চ ত্রিয়েট করি কিছু কমান্ড দিয়ে। তারপর সেখানে আমরা আমাদের নতুন ফিচার টেস্ট করি। তারপর ভালো লাগলে সে ব্রাঞ্চ মেইন প্রোজেক্টের সাথে মার্জ করে ফেলি, আর ভালো না লাগলে সে ব্রাঞ্চ থেকে আবার মেইন প্রোজেক্টে চলে আসি। এক্ষেত্রে আমরা যেহেতু অন্য ব্রাঞ্চে কাজ করেছি, তাই মেইন প্রোজেক্টে কোনো হাতই দেওয়া হয় নাই। খুব সহজেই কয়েকটা কমান্ড দিয়েই আবার মেইন প্রোজেক্টে চলে আসতে পারবো। আর ম্যানুয়ালী আমাদের কোড মুছে ফেলা বা ফাইল ডিলেট করা কিছুই করা লাগবে না।

# গিটহাব কি?

গিটহাব ব্যাসিকলি হোস্টিং সার্ভিস, তবে একটু স্পেশাল। কেমন স্পেশাল? হ্যাঁ ঠিক অনুমান করতে পেরেছেন, এটা গিট ভার্শন কন্ট্রোল সিস্টেমের জন্যে হোস্ট প্রোভাইড করে। আর সাথে কিছু ইউজার ইন্টারফেসও প্রোভাইড করে গিটের কাজগুলো করার জন্যে। এখন গিটহাবই একমাত্র হোস্ট প্রোভাইডার না এখানে, আরো যেমন বিটবাকেট, গিটল্যাবসহ আরো অনেক আছে।

কিন্তু আমি এখানে গিটহাব হাইলাইট করেছি। কারণ গিটহাবেই অনেক বড় বড় ওপেন সোর্স অনেক প্রোজেক্ট রয়েছে আর এটাই বেশী পপুলার। আর এদের সবার ইন্টারফেসেই কাছাকাছি, একটা শিখে ফেললে অন্যান্যগুলোতেও আপনি সহজেই কাজ করতে পারবেন। গিটহাবের অল্টারনেটিভ হিসেবে বিটবাকেট, গিটল্যাব থেকে শুরু

করে অসংখ্য ছোটো বড় এরকম সার্ভিস এভেইলেবল আছে, বিভিন্ন টিম বিভিন্ন সুবিধা-অসুবিধার কারণে একটার উপর আরেকটা চয়েজ করে।



গিট ও গিটহাব ব্যবহার করে কোলাবরেশন

এখন এগুলোতে আমার প্রোজেক্ট হোস্ট করে কি লাভ? হ্যাঁ ঠিক, সেটাই এখন খুলে বলবো কেন আমাদের এজনে হোস্টিং প্রোভাইডারও লাগবে। আমাদের প্রোজেক্টে মাঝেমধ্যে একাধিক ডেভেলপার থাকে বা টিমওয়ার্ক করলে একাধিক ডেভেলপাররা একই প্রোজেক্টের উপর কাজ করে। এখন আমরা চাইলে আমাদের প্রোজেক্টের ভার্শন কন্ট্রোল সিস্টেমসহ আমাদের বাকি টিমমেটদের সাথে শেয়ার করতে পারবো এইধরনের হোস্ট ব্যবহার করে, বা গিটহাব/বিটবাকেট/গিটল্যাব দিয়ে।

আমার টিমমেটৱাও আমার সেইম প্রোজেক্ট পাবে, চাইলে আমার আগের ভার্শনগুলোও দেখতে পারবে, আবার চাইলে নিজেও নতুন আরেকটা ভার্শন অ্যাড করে আবার গিটহাবের মাধ্যমে শেয়ার করতে পারবে। আর আমি আবার গিটে কমান্ড দিয়ে সে ভার্শন আমার লোকাল ডিস্কে নিয়ে আসতে পারবো।

এখন গিটহাবে প্রোজেক্ট এভাবে ওপেনও রাখা যায় আবার চাইলে প্রাইভেট প্রোজেক্টও রাখা যায়। গিটহাবে এমন অনেক প্রোজেক্ট দেখবেন যেখানে কয়েক হাজার ডেভেলপার কন্ট্রিবিউট করেছে একইসাথে, একই প্রোজেক্টে। এগুলো সবই সম্ভব হয়েছে মূলত এই গিট ও গিটহাবের মতো সার্ভিসের কারণে।

facebook/react Public

Code Issues 873 Pull requests 241 Actions Projects Wiki Security Insights

main 88 branches 141 tags

Go to file Code About

A declarative, efficient, and flexible JavaScript library for building user interfaces.

reactjs.org

react javascript library ui frontend declarative

Readme MIT license Code of conduct Security policy 199k stars 6.6k watching 41.3k forks

etoclear and epsilon [Float] Fix typo in ReactDOMResourceValid... 8196872 10 hours ago 15,423 commits

.circleci Remove Reconciler fork (1/2) (#25774) 6 days ago

.codesandbox Update Node.js to latest v14.17.6 LTS (#22401) 15 months ago

.github [DiffTrain] Add REVISION and REVISION\_TRANSFORM to output (#... 2 days ago

.fixtures Bump qs from 6.5.2 to 6.5.3 in /fixtures/concurrent/time-slicing (#... 23 hours ago

.packages [Float] Fix typo in ReactDOMResourceValidation.js (#25798) 10 hours ago

.scripts Revert "[react-www] remove forked bundle" (#25837) 2 days ago

.editorconfig https link to editorconfig.org (#18421) 3 years ago

.eslintignore Renamed packages/react-devtools-scheduling-profiler to package... 13 months ago

.eslintrc.js Remove Reconciler fork (1/2) (#25774) 6 days ago

রিআর্টে জেএস এর গিটহাব প্রোজেক্ট [রিপো](#)

গিট

# গিট সেটআপ

গিট ব্যবহার করতে চাইলে অবশ্যই আপনাকে গিট [এখান](#)(git-scm.com) থেকে ডাউনলোড করে ইন্সটল করতে হবে আপনার সিস্টেমে। আপনার অপারেটিং সিস্টেম যেটাই হউক না কেন, সবার জন্যই গিট এভেইলেবল।

সেটাপ প্রসেসে আমি বিস্তারিত ঘাবো না। খুবই সিম্পল, যদি কিছু বুঝতেও না পারেন জাস্ট নেক্স্ট নেক্স্ট দিয়ে সেটাপ প্রসেস কমপ্লিক্ট করুন। সেটাপ করা শেষ হলে একটা গিট ব্যাশ (Git Bash) অ্যাপ্লিকেশন পাবেন। এটা কমান্ড লাইন এনভারোমেন্ট। এটা ওপেন করলে কমান্ড দেওয়ার উইন্ডো পাবেন, এখানে আপনি ইউনিক্স-লাইক অপারেটিং সিস্টেমের কমান্ড ব্যবহার করতে পারবেন। এজন্যে

আপনার আগের কিছু লিনাক্স/ইউনিক্স এর কমান্ডের সাথে পরিচয় থাকলে সহজেই এখানে ব্যবহার করতে পারবেন।

অথবা আজকে এখানে যে যে কমান্ডগুলো ব্যবহার করব সেগুলো কোনটা কিভাবে কাজ করে সেগুলো শিখে ফেললেই আপাতত আপনি গিট ব্যবহার করতে পারবেন। এখন আপনি চাইলে আপনার কম্পিউটারে থাকা সব কমান্ড লাইন/টার্মিনাল থেকেই এখন গিট চালাতে পারবেন। আপনার পছন্দের কমান্ড লাইন/টার্মিনাল ওপেন করে নিচের কমান্ডটি লিখুনঃ

```
> git --version
```

এটা এরকম কিছু আউটপুটে আপনার গিটের ভার্শন দেখাবে। ভার্শন নাম্বার অবশ্যই আমার দেখানোটার চেয়ে ডিফারেন্ট হবে, কারণ আমারটা আমি যখন এই বই লিখি তখনকার ভার্শন দেখাচ্ছঃ

```
> git version 2.30.1 (Apple Git-130)
```

# গিট কনফিগার

গিটের প্লোবাল কিছু কনফিগারেশন করে নিতে হবে সবকিছু শুরু করার আগে। খুবই সিম্পল। জাস্ট আপনার কমান্ড লাইনটা ওপেন করে নিচের কমান্ডগুলো নিজের নাম এবং ইমেইল দিয়ে সেটাপ করে নিন।

নিচের কমান্ডগুলো গিট এর প্লোবাল কনফিগারেশন। অর্থাৎ আপনার সিস্টেমে যত প্রোজেক্টে গিট ব্যবহার করবেন তার সবগুলোতে ইউজারের নাম আর ইমেইল হিসাবে এগুলোই ব্যবহার করবে।

```
> git config --global user.name "Zonayed Ahmed"  
> git config --global user.email "zonayedpca@yahoo.com"
```

ঠিক এভাবেই আপনি আপনার নাম আর আপনার ইমেইল দিবেন এখানে। ব্যাস কাজ কমপ্লিট।

কিন্তু আপনি যদি চান একাধিক প্রজেক্টের জন্য একাধিক নাম ও ইমেইল থাকবে তাহলে global কীওয়ার্ড ও তার আগের হাইফেন দুটো কেটে দিন। যেমন আপনার পিসিতে অফিসের একটা প্রজেক্ট আছে আবার আপনার পার্সোনাল একটা প্রজেক্ট আছে। অফিসের প্রজেক্টটি রাখা আছে গিটহাবের আপনার অফিসের অ্যাকাউন্টে। আপনি সেই অ্যাকাউন্টে অ্যাক্সেস করেন আপনার অফিসের ইমেইল দিয়ে। আর আপনার পার্সোনাল গিটহাব অ্যাকাউন্ট খোলা হয়েছিল আপনার পার্সোনাল ইমেইল দিয়ে। তাহলে আপনার প্রজেক্টে যদি প্লেবাল ইউজার নেম আর ইমেইল সেট করা থাকে তখন কিন্তু সব প্রজেক্টেই আপনার একই নাম ও একই ইমেইল দেখাবে। এজন্য প্লেবাল ইউজারনেম, ইমেইলের পাশাপাশি কোন প্রোজেক্টে অন্য কোন ইউজারনেম, ইমেইল ব্যবহার করতে চাইলে উক্ত প্রোজেক্টে গিট ইনিশিয়ালাইজ করার পর এভাবে কমান্ড ব্যবহার করে শুধুমাত্র সেই প্রোজেক্টের জন্য ইউজারনেম ও ইমেইল সেটআপ করতে পারবেনঃ

```
> git config user.name "Zonayed Ahmed"  
> git config user.email "zonayedpca@yahoo.com"
```

# গিট রেপোজিটরি সেটআপ

গিটে ডিরেষ্টরিকেই ব্যাসকালি রিপোজিটরি (Repository) বা শর্টকাটে অনেকে ‘রিপো (Repo)’ বলে। আপনার অলরেডি প্রোজেক্ট আছে এমন কোনো প্রোজেক্টে গিট স্টার্ট করতে চাইলে প্রথমে আপনার গিট ব্যশ বা আপনার যেকোনো কমান্ড লাইন থেকে সে প্রোজেক্টের ডিরেষ্টরিতে যেতে হবে। সে ক্ষেত্রে আপনি যদি গিটের সেটাপের সময় কোনো অপশন পরিবর্তন করে না থাকেন তাহলে আপনার প্রোজেক্টের ভিতরে রাইট লিঙ্ক করলে দেখবেন Git Bash Here(বিশেষ করে Windows অপারেটিং সিস্টেমে) নামে একটা অপশন দেখাবে। এটাতে আপনার কান্সিফিত প্রোজেক্ট ডিরেষ্টরির ভিতর থেকে লিঙ্ক করলে এই ডিরেষ্টরিতে গিট ব্যশ ওপেন হবে যেখানে আপনি কমান্ড লিখতে পারবেন।

এখন ধরি আপনার Desktop এ একটা ডিরেক্টরি আছে learning-git নামে(আপনি চাইলে কমান্ড লাইনের সাহায্যে কমান্ড দিয়েও এই ডিরেক্টরিটা তৈরি করে নিতে পারেন অথবা ইউআর্ড ব্যবহার করেও নতুন ডিরেক্টরি/ফোল্ডার তৈরি করে নিতে পারেন এই নামে)। আর এই ডিরেক্টরি/ফোল্ডারের ভিতরে কিছু ফাইল রাখবো আমরা friend-list.txt আর QnA.txt নামে।

ধরি, friend-list.txt ফাইলের ভিতরে কন্টেন্ট আছে এরকমঃ

Dibakar Sutradhar  
S M Shahadat Hossain  
Reduanul Houque Munna  
Ar Rolin  
Niraj Paudel  
Tanvir Faisal Moon  
Sagar Neupane  
Yadav Lamechane

আর QnA.txt ফাইলের ভিতরে আপাতত কিছু রাখার দরকার নাই, এটা খালিই রাখতে পারেন। আমরা আপাতত আমাদের friend-list.txt ফাইলটা নিয়েই কাজ করব।

এখন এটাই আপনার প্রোজেক্ট, এখানেই আমরা গিট ইনেশিয়েলাইজ করতে চাই। তাহলে আমি আমার কমান্ড লাইন এই ডিরেক্টরির ওপেন করে নিচের এই কমান্ড লিখবোঃ

```
> git init
```

এরকম আউটপুট দেখবেন

```
> Initialized empty Git repository in /Users/zonayedpca/Desktop/learning-git/.git/
```

আমি learning-git ডিরেক্টরির ভিতর থেকে git init কমান্ড রান করলাম যেহেতু আমি এটার ভিতরের সবকিছুই ট্র্যাক করতে চাই। ব্যাস এখন এই ডিরেক্টরির ভিতরে গিটের রিপো সেটাপ হয়ে গেলো। এখন থেকে গিট সব ট্র্যাক করা শুরু করতে পারবে, এই ডিরেক্টরির ভিতরে যতো ফাইল/ফোল্ডার আছে সব। তবে ট্র্যাক করলেও গিট সেগুলোকে ভার্শন হিসাবে স্টোর করবে না। তারজন্যে আপনাকেই স্পেসেফিকলি বলে দিতে হবে কোনটা কোনটা কখন কিভাবে সেইভ করতে হবে।



কাজের ফ্লো

আমার এই ডিরেক্টরির ভিতরে দুইটা .txt ফাইল আছে। এগুলো এখন আমি চাছি গিট ভার্শন হিসাবে সেইভ করে রাখুক। তারজন্যে আমাদের দুইটা স্টেজ ক্রস করতে হবে। প্রথমে গিট আপনার উল্লেখিত ফাইলকে স্টেজিং এরিয়াতে নিবে, তারপর আবার আপনি চাইলে সেটা ফাইনাল হিসাবে আপনার গিট রিপোতে কমিট করতে পারবেন। এই দুই স্টেজের জন্য পৃথক পৃথক দুইটা কমান্ড ব্যবহার করতে হবে(অথবা সিঙ্গেল কমান্ড ব্যবহার করেও করা যাবে, কিন্তু শিখার সুবিধার্থে আমরা এখানে আলাদা আলাদা কমান্ড দিয়েই কাজ করব)।

# স্ট্যাটাস চেক করা

তার আগে আমরা গিটের বর্তমান অবস্থা দেখতে চাছি, মানে বর্তমান স্ট্যাটাস দেখতে চাছি কোন কোন ফাইল ট্র্যাক করা হয় নি বা কোন ফাইল স্টেজিং এ আছে। সেজন্যে নিচের এই কমান্ড ব্যবহার করতে হবেঃ

```
> git status
```

আমার এই ডিরেক্টরিতে দুইটা ফাইল আছে QnA.txt আর friend-list.txt নামে। আমি যেহেতু মাত্রই গিট ইনিশিয়েট করলাম এই প্রোজেক্টে তাই দুইটা ফাইলই এখানে আন-ট্র্যাকড দেখাচ্ছে। আর সাথে কিছু হিন্টও দিয়ে দিচ্ছে কিভাবে ফাইলগুলো ট্র্যাক করতে হবে।

> On branch main

No commits yet

Untracked files:

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

QnA.txt

friend-list.txt

nothing added to commit but untracked files present (use

# স্টেজিং এরিয়াতে নেওয়া

আমি প্রথম QnA.txt ফাইলটা ট্র্যাক করতে চাই বা যেটাকে বলে স্টেজিং এরিয়াতে নিতে চাই। সেজন্যে আমাকে এভাবে কমান্ড দিতে হবেঃ

```
> git add QnA.txt
```

এখন আপনার ফাইল কোনো ডিরেক্টরির ভিতরে হলে তাহলে সেভাবে ফাইলের রেফারেন্স দিতে হতো git add <Your file> এভাবে। এখন আবার git status দিলে দেখবেন বর্তমান স্ট্যাটাসঃ

```
> git status
```

এখানে এখন দুইটা সেকশন দেখাচ্ছে। যেটা ট্র্যাক করেছি সেটা এখন  
উপরে দেখাচ্ছে Changes to be committed সেকশন এ। আর  
নিচে আগের সেই আন-ট্র্যাকড ফাইলটাই দেখাচ্ছে:

```
> On branch main
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)  
  new file:   QnA.txt
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be  
committed)  
  friend-list.txt
```

যাই হোক এই মুহূর্তে আপনার ফাইল QnA.txt স্টেজিং এরিয়াতে  
আছে, তাই এখন কমিট করলে গিট শুধুমাত্র এই ফাইলটাকেই ভার্শন

হিস্টোরীতে রাখবে। আর যেটা এখনো ট্র্যাক করা হয় নাই সেটাকে নিয়ে কিছু করবে না। এখন যদি আমরা চাই যে এই ডিরেষ্টরির ভিতরের সব আন-ট্র্যাকড ফাইলকে ট্র্যাক করতে একটা কমান্ড দিয়ে তাহলে এভাবে দিতে হবেঃ

```
> git add --all
```

অথবাঃ

```
> git add .
```

এখন `git status` দিলে দেখবেন সব ট্র্যাক হয়ে গেছে, মানে স্টেজিং এরিয়াতে আছে। কোনো ফাইল আন-ট্র্যাকড নাই। আগের `QnA.txt` এখনো স্টেজিং এ আছে, যেহেতু এটা আমরা এখনো কমিট করি নাই। আর সাথে এখন `friend-lists.txt` ও চলে আসছে। এখন কমিট করলে দুইটা মিলেই পুরোটার একটা ভার্শন রাখবে গিট।

# ফাইনাল কমিট করা

০৫

কমিট হচ্ছে আপনি ফাইনাল সিন্ধান্ত নিবেন আপনার ট্র্যাক করা চেঞ্জেসগুলোকে গিট রিপোতে রাখতে। এখন কমিট করতে চাইলে, প্রত্যেক কমিটের সাথে একটা ম্যাসেজও দিতে হয় যাতে পরবর্তিতে একদিন পরে বা এক বছর পরে বুঝতে সুবিধা হয় অমুক কমিটটা কি কারণে করা হয়েছিলো। সবকিছু এক লাইনে এভাবে হবেঃ

```
> git commit -m "QnA and Friend Lists Added"
```

এখানে QnA and Friend Lists Added হচ্ছে আমাদের এই কমিটের ম্যাসেজ। কমান্ড দেওয়া হলে এরকম ম্যাসেজ দেখতে পাবেনঃ

```
> [main (root-commit) 7810dd3] QnA and Friend Lists Added
  2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 QnA.txt
create mode 100644 friend-list.txt
```

---

আপনি স্টেজিং এরিয়ারে নেওয়ার আগে কোনোকিছু কমিট করতে  
পারবেন না।

---

# ফাইল মডিফার করে আবার কমিট

এখন আমার একটা ভার্শন তৈরী হয়ে গেলো। কিন্তু আমার প্রোজেক্টে আরো অনেক কাজ আছে। আমি চাচ্ছি friend-lists.txt ফাইলে সব ফ্রেণ্ডদের ফোন নাম্বার সেইভ করে রাখতে। বর্তমানে ফাইলটা এরকম আছেঃ

Dibakar Sutradhar

S M Shahadat Hossain

Reduanul Houque Munna

Ar Rolin

Niraj Paudel

Tanvir Faisal Moon

Sagar Neupane

Yadav Lamechane

এখানে এই টেক্সট ফাইল এডিট করা আর কোনো কোড এডিট করা  
একই কথা। আমি টেক্সট ফাইলের সাহায্যে দেখাচ্ছি যাতে সবার  
বুকতে সুবিধা হয়। এখন আমি সবার সাথে ফোন নাম্বার অ্যাড করবঃ

Dibakar Sutradhar - +88018XXXXXXX

S M Shahadat Hossain - +88018XXXXXXX

Reduanul Houque Munna - +88018XXXXXXX

Ar Rolin - +88018XXXXXXX

Niraj Paudel - +9718XXXXXXX

Tanvir Faisal Moon - +88018XXXXXXX

Sagar Neupane - +9718XXXXXXX

Yadav Lamechane - +9718XXXXXXX

এখন টেক্সট এডিটর বা কোড এডিটর যেটাই ব্যবহার করে ফাইল  
মডিফাই করলাম সেটাতে সেইভ দিয়ে git status চেক করলে  
দেখবেন ফাইল এটা মডিফাইড দেখাচ্ছেঃ

> **git status**

এখানে মূলত বলা হচ্ছে আপনি যে পরবর্তনগুলো করেছেন সেগুলো  
কমিট করার জন্য স্টেজড করা হয়নি, বা ফাইনাল কমিটের জন্য  
স্টেজিং এরিয়াতে নেওয়া হয়নিঃ

```
> On branch main  
Changes not staged for commit:  
  (use "git add <file>..." to update what will...  
  (use "git restore <file>..." to discard changes...  
    modified:   friend-list.txt  
no changes...commit (use "git add" and/or "git commit -a")
```

এখন এই আন-ট্র্যাকড ফাইলটাকে স্টেজিং এ নিয়ে ফাইনাল কমিট করে দিতে চাছিঃ

```
> git add --all
```

এবং ফাইনাল কমিটের জন্যঃ

```
> git commit --m "Contact Numbers Added"  
> [main xxxxxxx] Contact Numbers Added  
  1 file changed, 8 insertions(+)
```

ব্যাস কমান্ড দেওয়ার পরে নিচে উপরের মতো এরকম আউটপুট  
দেখতে পাবেন। কোন এরর বা ভুল হলে অবশ্যই অন্যরকম আউটপুট  
আসবে, তাই পড়ে শিউর হয়ে নিবেন যে আসলে কি হয়েছে।

---

বই থেকে কোড কপি না করে ম্যানুয়ালি হাতে লেখার চেষ্টা করবেন।  
বই থেকে যদি কপি করেও থাকেন তাহলে একটু শিউর হয়ে নিবেন  
সবগুলো ঠিকঠাক কপি হয়েছে কিনা। এই যেমন ডাবল কোটেশন  
মার্কগুলো ("...") কপি করলে সেটা ঠিকঠাক কাজ করে না,  
সেক্ষেত্রে আপনাকে নিজে ডাবল কোটেশন মার্ক ম্যানুয়ালি লিখে  
ইনপুট দিতে হতে পারে।

---

# পুনরায় মডিফাই করে কমিট

09

এতক্ষণ ধরে হয়তো বিভিন্ন জায়গায় নিশ্চয়ই একটা লেখা দেখেছেন।  
বিভিন্ন জায়গায়। সামথিং `main` টাইপের কিজানি লিখা উঠে। অথবা  
যদি আপনি `git status` কমান্ডটি ব্যবহার করেন তাহলে  
শুরুতেই এরকম(On branch ...) একটা লেখা পাবেনঃ

```
> git status
> On branch main ...
```

এই `main` হলো বর্তমান ভাস্কের নাম। অর্থাৎ বর্তমানে আপনি মেইন  
ভাস্ক বা বর্তমান ওয়ার্কিং ডিরেক্টরিতে আছেন। এটাই আপনার  
প্রোজেক্টের বর্তমান ভার্শন।

এখন আমাদের ডেভেলপমেন্ট এ অনেক সময় দেখা যায় পূর্বের ভার্মনে ফিরে যেতে হয়। একটা একটা করে ফিচার ডেভেলপড করার পর একটা সময় এসে কোনো প্রেরণ দেখা দেয় যেটা পূর্বের কোনো ভার্মনে ঠিকঠাক কাজ করতে কিন্তু এখন সেটা কাজ করছে না। সেক্ষেত্রে গিট এ ট্র্যাক করা থাকলে আপনি সহজেই আপনার সেই ভার্মনে ফিরে যেতে পারবেন আর কোড চেক করতে পারবেন, চাইলে আপনার প্রোজেক্ট রানও করতে পারবেন। ঠিক ঐসময় আপনার প্রোজেক্ট ঘেরকম ছিলো সেরকমটাই দেখাবেন।

আমরা এখন ইচ্ছাকৃতভাবেই friend-lists.txt ভিতরে অ্যাড করা ফোন নাম্বারগুলো মুছে দিয়ে কমিট করব আরেকটা। মুছে ফেলার পর ফাইলটা এরকম হবেঃ

Dibakar Sutradhar

S M Shahadat Hossain

Reduanul Houque Munna

Ar Rolin

Niraj Paudel

Tanvir Faisal Moon

Sagar Neupane

Yadav Lamechane

এখন এটা সেইভ করে স্টেজিং এ অ্যাড করে কমিট করে দিবোঃ

```
> git add -all
```

এবার কমিট ম্যাসেজ লিখে কমিট করুনঃ

```
> git commit --m "Contact numbers removed"  
> [main xxxxxxx] Contact numbers removed  
 1 file changed, 8 insertions(+), * deletions(-)
```

০৮

# কমিট লগ চেক

হায় হায়! এটা কি হলো!! ফোন নাম্বার সব গেলো!!! কি হবে এখন? হ্যাঁ গিট দিয়ে তো ট্র্যাক করেই রেখেছি সব। কোন কমিটে জানি ফোন নাম্বারগুলো রেখেছিলাম? হ্যাঁ সেটা দেখতে চাচ্ছি। সব কমিটের লগ দেখতে চাইলৈঃ

```
> git log
```

এখানে তিনটা কমিট আছে। সাথে ডিটেইলস সহ, কমিটের ম্যাসেজ দেখে সহজেই বুঝতে পারবেন কোন কমিটে কি করা হয়েছিলো। আর সাথে কিছু এলোমেলো নাম্বার আছে। এগুলো ব্যবহার করে আমরা পূর্বের ভার্শনগুলোয় ফিরে যেতে পারবোঃ

> commit 34432c0a1fabb801ae...3c4358f60d4a1 (HEAD -> main)

Author: Zonayed Ahmed <zonayedpca@yahoo.com>

Date: Sun Dec 11 12:10:41 2022 +0600

Contact numbers removed

commit fac6322f14171b250a15f888ccf3d2874f7cff0d

Author: Zonayed Ahmed <zonayedpca@yahoo.com>

Date: Sun Dec 11 11:48:11 2022 +0600

Contact Numbers Added

commit 7810dd39b0e59af3d9c40151462a9655a4008470

Author: Zonayed Ahmed <zonayedpca@yahoo.com>

Date: Thu Dec 8 22:15:52 2022 +0600

QnA and Friend Lists Added

---

বিঃদ্রঃ এই অবস্থা স্থিতে আরো কমিট (যদি থাকে) দেখতে কীবোর্ডের আপ-ডাউন কীগুলো ব্যবহার করতে পারেন। আর এখান থেকে বের হতে **q** বাটন চাপ দিলেই হবে।

---

এই লগ টা আরো সুন্দর করে কম্প্যাক ভার্শনে দেখতে চাইলে উপরের কমান্ডটা এভাবেও দেওয়া যাবেং

```
> git log --oneline
```

এখানে সুন্দর করে ছোটো করে প্রয়োজনীয় সব দেখাচ্ছে। এখন এইখানের শর্টকাট এলোমেলো ইউনিক কমিট আইডিগুলোও শর্ট করে দেওয়া হয়েছে। এই শর্ট ভার্ণনগুলোও ব্যবহার করে আগের কাঞ্চিত ভার্ণনে যেতে পারবেনঃ

```
> 34432c0 (HEAD -> main) Contact numbers removed  
  fac6322 Contact Numbers Added  
  7810dd3 QnA and Friend Lists Added
```

# পূর্বের ভার্শনে যাওয়া

০৯

এখন আমরা যে কমিটে ফোন নাম্বার গুলো অ্যাড করেছিলাম সে কমিটে ফিরে যেতে চাচ্ছি। আমার এখানে সেই কমিটটা হলো এটাঃ

> `fac6322 Contact Numbers Added`

এখন এই ভার্শনে ফিরে যেতে চাইলে গিটের আরেকটা কমান্ড এইভাবে ব্যবহার করতে হবেঃ

> `git checkout fac6322`

এখানে শেষেরটা হচ্ছে কমিট আইডি। আপনার আইডি ভিন্ন হবে। এখন এই কমান্ড রান করলে আপনার প্রোজেক্ট main ব্রাঞ্চ থেকে আগের এই কমিটের ভার্সনে ফিরে যাবে। তবে অবশ্যই মাস্টার ব্রাঞ্চে থাকাকালে সবকিছু আপনার ট্র্যাক করা থাকতে হবে। কোনো ফাইল/ফোল্ডার আন-ট্র্যাকড থাকলে বা আন-কমিটেড থাকলে আপনি চেক-আউট করতে পারবেন না। এখন কমান্ড লাইনে main এর জায়গায় কমিট আইডিটা দেখবেন। সাথে দেখবেন লেখা HEAD detached at আপনার কমিট আইডি।

> Note: switching to 'fac6322'.

You are in 'detached HEAD' state... make experimental changes and commit them... discard any commits you make in this state without impacting an... switching back to a branch.

If you want to create a new branch... you create, you may do so (now or later) by using -c with the switch command.

Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable  
advice.detachedHead to false

```
HEAD is now at fac6322 Contact Numbers Added
```

এখন আপনার ফাইল চেক করে দেখুন আগের সেই ভার্শনে ফিরে আসছে। friend-lists.txt তে সবার ফোন নাম্বারগুলো রয়েছেঃ

Dibakar Sutradhar - +88018XXXXXXX  
S M Shahadat Hossain - +88018XXXXXXX  
Reduanul Houque Munna - +88018XXXXXXX  
Ar Rolin - +88018XXXXXXX  
Niraj Paudel - +9718XXXXXXX  
Tanvir Faisal Moon - +88018XXXXXXX  
Sagar Neupane - +9718XXXXXXX  
Yadav Lamechane - +9718XXXXXXX

এখন আপনার বর্তমান ওয়ার্কিং ডিরেক্টরি আগের একটা ভার্শনে রয়েছে। কিন্তু আপনি যদি মেইন ভার্শনে যেতে চান তাহলে আবার চেক-আউট দিতে হবে এভাবেঃ

```
> git checkout main
> Previous HEAD position was fac6322 Contact Numbers Added
Switched to branch 'main'
```

# ব্রাঞ্চ তৈরি

৩০

আমি আগেই ব্রাঞ্চ (branch) এর কথা বলেছিলাম। তবে ব্রাঞ্চ কে আরো স্পেসেফিকলি বললেং- ব্রাঞ্চ আসলে আপনার করা কমিটগুলোই, কিন্তু সেই কমিটগুলোর একটা ইউনিক নাম থাকবে। আপনি সেই কমিটে চেক-আউট করতে চাইলে ব্রাঞ্চ এর নাম দিয়েই চেক-আউট করতে পারবেন। আগের সেই আশ্চর্য টাইপের কমিট আইডি লাগবে না।

আমি এখন আমার এই প্রোজেক্টে নতুন কিছু ট্রাই করতে চাচ্ছি। তবে আমি মেইন প্রোজেক্টে বা মেইন ব্রাঞ্চে সেটা এখনি আনতে চাচ্ছি না। বলতে পারেন আমি এখন এক্সপেরিমেন্টাল কিছু একটা করব। তারপর ভালো লাগলে মেইন ব্রাঞ্চে নিয়ে আসবো।

এইজন্যে আমরা নতুন একটা ব্রাঞ্চ তৈরী করব `table-version` নাম দিয়েং।

```
> git branch table-version
```

এখন আপনার এই `table-version` নামে একটা ব্রাঞ্চ তৈরী হয়ে যাবে। আপনি যে ব্রাঞ্চ থেকে এই নতুন ব্রাঞ্চ তৈরী করবেন, নতুন ব্রাঞ্চে সেই ভার্শনটাই থাকবে। আমার ক্ষেত্রে আমি `main` ব্রাঞ্চ থেকে `table-version` ব্রাঞ্চ তৈরী করেছি। আর তাই `table-version` এ আমার বর্তমানে `main` ব্রাঞ্চ এ থাকা প্রোজেক্টের ভার্শনটাই যাবে। মানে এখন `main` আর `table-version` এর প্রোজেক্ট পুরোপুরি সেইম।

আপনি চাইলে আপনার প্রোজেক্টে থাকা সবগুলা ব্রাঞ্চ এর লিস্ট ও দেখতে পারবেন (এখান থেকে বের হতে `q` চাপুন):

```
> git branch  
> main  
  table-version
```

# ବ୍ରାଂସ ଏ ଚେକଆଡ଼ଟ

ଆମରା ବ୍ରାଂସ ତୈରୀ କରେଛି, କିନ୍ତୁ ସେଇ ବ୍ରାଂସ ଏ ଏଥିରେ ଚେକ-ଆଡ଼ଟ କରିନି। କୋଣ ବ୍ରାଂସ ଏ ଆଛି ତା ଆପନାର କମାନ୍ ଲାଇନେ କାରେନ୍ଟ ଓୟାର୍କିଂ ଡିରେଷ୍ଟରିର ପାଶେ ଦେଖିଲେଇ ବୁଝାବେନ, ଅଥବା `git status` ବ୍ୟବହାର କରେଓ ଦେଖିତେ ପାରେନ। ଆମରା ଆମାଦେର ପ୍ରୋଜେକ୍ଟେ ଏଥିରେ `main` ବ୍ରାଂସେଇ ଆଛି।

- > `git status`
- > On branch main...

ଏଥିରେ ନତୁନ କ୍ରିଯେଟ କରା ବ୍ରାଂସେ ଚେକ-ଆଡ଼ଟ କରା ଠିକ ଆଗେର ଅନ୍ୟ କୋନୋ କମିଟେ ଚେକ-ଆଡ଼ଟ କରାର ମତୋହି। ଶୁଧୁମାତ୍ର ଏକେବେଳେ ଆମରା ବ୍ରାଂସ ଏର ନାମ ଦିଯେଇ ଚେକ-ଆଡ଼ଟ କରତେ ପାରିବାଃ

```
> git checkout table-version
```

এখন দেখবেন আপনার ব্রাঞ্চে table-version এ চলে গেছে। এখানেও একটা ছোট্ট শর্টকাট টেকনিক আছে। আপনি যদি চান নতুন ব্রাঞ্চে তৈরী করে সাথে সাথে সেই ব্রাঞ্চে চেক-আউট করতে, সেটা একলাইনের কমান্ডেই করতে পারবেনঃ

```
> git checkout -b table-version-new
```

দেখুন আমরা নতুন একটা ব্রাঞ্চে table-version-new নামে তৈরী করেছি এবং সাথে সাথে সেই ব্রাঞ্চে চেক-আউট করে ফেলেছি।

যাই হোক এখন আমরা table-version এ কিছু মডিফাই করে তারপর সেগুলো মেইনে মার্জ করব। তাই git checkout table-version দিয়ে আমরা আমাদের কাঞ্চিত ব্রাঞ্চে চলে যাবো। অবশ্যই কাজ করার সময় খেয়াল করবেন কোন ব্রাঞ্চে আছেন। কষ্ট করে কারেন্ট ওয়ার্কিং ডিরেক্টরির ডান পাশে দেখলেই পাবেন কোন ব্রাঞ্চে আছেন সেটা।

# নতুন ব্রাঞ্জে মডিফিকেশন

১৮

এখন আমরা আমাদের এই নতুন table-version ব্রাঞ্জে নতুন  
কিছু ট্রাই করব। বর্তমানে আমাদের প্রোজেক্টের friend-  
lists.txt ফাইল এই অবস্থায় আছেং

Dibakar Sutradhar

S M Shahadat Hossain

Reduanul Houque Munna

Ar Rolin

Niraj Paudel

Tanvir Faisal Moon

Sagar Neupane

Yadav Lamechane

এখন আমরা এই নামগুলো একটা টেবিলের ভিতরে নিয়ে দেখি কেমন  
লাগেঃ

```
=====
|| Dibakar Sutradhar      ||
=====
|| S M Shahadat Hossain  ||
=====
|| Reduanul Houque Munna ||
=====
|| Ar Rolin                ||
=====
|| Niraj Paudel            ||
=====
|| Tanvir Faisal Moon     ||
=====
|| Sagar Neupane           ||
=====
|| Yadav Lamechane         ||
=====
```

ধরে নিলাম আমার কাজের এই ভার্শনটা আমার ভালো লেগেছে, এখন  
আমি এটা মেইন ব্রাউজে বা মেইন প্রোজেক্টে নিয়ে যেতে চাই। কিন্তু  
তার আগে আপনার এই পরিবর্তনগুলো বর্তমান ব্রাউজে কমিট করতে

হবে। কারণ আপনি যতক্ষণ পর্যন্ত কোনো কিছু কমিট না করবেন, গিট সেগুলোকে কাউন্টই করবে না। কমিট করার জন্যঃ

```
> git add --all  
> git commit -m "Table added"
```

ব্যাস কমিট হয়ে গেলো আমার নতুন পরিবর্তনগুলোঃ

```
> [table-version 1a9c516] Table added  
  1 file changed, 17 insertions(+), 8 deletions(-)
```

এখন আমি এই `table-version` এ থাকা কাজগুলো `main` ভাঁধে নিতে চাই। সেজন্যে আমাদেরকে প্রথমে `main` ভাঁধে চেক-আউট করতে হবে এভাবেঃ

```
> git checkout main
```

ভাঁধের নাম যেহেতু `main`, তাই এটা লিখেই চেক-আউট করতে পারবেন। এখন খেয়াল করুন আপনার `main` ভাঁধে যাওয়ার পর

আপনার প্রোজেক্টের সেই আগের ভার্শনটাই রয়ে গেছে।  
নতুন `table-version` এ করা কাজ এখানে আসে নাই। আপনি  
যদি `table-version` এ করা কাজ ফেলে দিতে চাইতেন,  
তাহলে জাস্ট `main` চেক-আউট করে চলে আসলেই হচ্ছে, কোথাও  
কোনো লেখা বা কোডে হাত দিতে হবে না।

মনে করি নতুন ব্রাঞ্চে করা কাজ আমার ভালো লাগে নাই, বা এটা  
আমি রাখতে চাচ্ছি না। তাহলে জাস্ট সেই ব্রাঞ্চটাকে এভয়েড  
করে `main` এ চেক-আউট দিলেই চলবে বা চাইলে ব্রাঞ্চ ডিলেটও  
করে দিতে পারবেন। তবে আমরা `table-version` টা রাখবো।  
কিন্তু এর সাথে কিন্তু আমরা আরেকটা ব্রাঞ্চ তৈরী  
করেছিলাম `table-version-new` নামে।

ব্রাঞ্চ এর লিস্ট দেখতেঃ

```
> git branch
> main
      table-version
      table-version-new
```

আমরা `table-version-new` ব্রাঞ্চ ডিলেট করব এখনঃ

```
> git branch -D table-version-new
```

এখন এই ব্রাঞ্চ ডিলেট হয়ে যাবে, আর সেই সাথে ব্রাঞ্চে কোনো মডিফিকেশন থাকলে সেগুলোও ডিলেট হয়ে যাবে।

> Deleted branch table-version-new (was 1a9c516).

# ବ୍ରାନ୍ଡ ମେହିନେ ମାର୍ଜ

୩୩

ଏখନ ମେହିନେ (main) ଚେକ-ଆଉଟ କରାର ପରେ ଦେଖିବେଳେ ମେହିନେ ଆଗେର ଭାର୍ଣ୍ଣନେଇ ଆଛେ। ଏଥିରେ ଆମରା table-version ଏ କରା ମଡ଼ିଫିକେଶନଗୁଲୋ ମେହିନେ ଆନତେ ଚାହିଁ। ସେଟା ଏକଦମ ସହଜ। main ବ୍ରାନ୍ଡେ ଥାକା ଅବସ୍ଥା ଏହି କମାନ୍ଡ ଦିଲେଇ ଅଟୋମ୍ୟାଟିକ ମାର୍ଜ ହେବେ:

```
> git merge table-version
> Updating 34432c0..1a9c516
Fast-forward
  friend-list.txt | 25 ++++++-----+
1 file changed, 17 insertions(+), 8 deletions(-)
```

সেই সাথে `table-version` এর কমিটটাও গিট অটোম্যাটিক  
অ্যাড করবে। গিট লগ দেখলে সেটাই দেখতে পাবেনঃ

```
> git log --oneline
>. 1a9c516 (HEAD -> master, table-version) Table added
  34432c0 Contact numbers removed
  fac6322 Contact Numbers Added
  7810dd3 QnA and Friend Lists Added
```

# কমিটের সাথে কমিটের পার্থক্য

এখন আমরা যদি আমাদের বর্তমানের কমিটের সাথে আগের কমিটের পার্থক্য দেখতে চাই, কী কী কোড পরিবর্তন হয়েছে, কোথায় কোড অ্যাড করা হয়েছে, কোথায় ডিলেট করা হয়েছে, এগুলোও সব দেখতে পারবো গিটের কমান্ডের সাহায্যে।

ধরি, আমরা Contact numbers removed আর Contact Numbers Added এই দুইটা কমিটের পার্থক্যগুলো দেখতে চাচ্ছি। তাহলে এই দুটোরই কমিট আইডি লাগবে। কমিট আইডি গিট লগ (git log অথবা git log --oneline) দিয়ে সহজেই বের করতে পারবেন। এখানে git diff এর সাথে উক্ত দুইটা কমিটের আইডি পাস করতে হবে এভাবেঃ

```
> git diff 34432c0 fac6322
> diff --git a/friend-list.txt b/friend-list.txt
--- a/friend-list.txt
+++ b/friend-list.txt
-Dibakar Sutradhar
-S M Shahadat Hossain
-Reduanul Houque Munna
-Ar Rolin
-Niraj Paudel
-Tanvir Faisal Moon
-Sagar Neupane
-Yadav Lamechane
+Dibakar Sutradhar - +88018XXXXXXX
+S M Shahadat Hossain - +88018XXXXXXX
+Reduanul Houque Munna - +88018XXXXXXX
+Ar Rolin - +88018XXXXXXX
+Niraj Paudel - +9718XXXXXXX
+Tanvir Faisal Moon - +88018XXXXXXX
+Sagar Neupane - +9718XXXXXXX
+Yadav Lamechane - +9718XXXXXXX
```

এখানে উক্ত দুইটা কমিটে কোন ফাইলে এবং ঠিক কি কি রিমুভ  
(লালগুলো) এবং অ্যাড(সবুজগুলো) করা হয়েছে সেগুলো দেখানো  
হচ্ছে।

এখানে আরো লক্ষ্য করুন আমি git diff এর সাথে প্রথম আর্গুমেন্ট এ মোস্ট রিসেন্ট কমিট এবং পরেরটায় সেই কমিটের আগের কমিটের আইডি দিয়েছি। মানে প্রথম নতুনটা আর পরে পুরোনোটা দিয়েছে। এটার মানে হচ্ছে আমি প্রথমটার সাথে দ্বিতীয়টার পার্থক্য দেখতে চাচ্ছি। এক্ষেত্রে দ্বিতীয়টা অর্থাৎ পুরোনোটার অনুসারে অ্যাডেড বা রিমুভড কোডগুলো দেখাবে। সেই সাথে কমিট আইডি প্রথমে পুরোনোটা এবং পরে নতুনটা দিলে ঠিক উল্টোটা দেখতে পাবেন। নতুনটার অনুসারে দেখাবে। কয়েকবার নিজে কমান্ড দিয়ে দেখলেই বুঝতে পারবেন।

---

বিঃদ্রঃ এই অবস্থা স্ক্রিনে আরো পার্থক্য দেখতে (যদি থাকে) কীবোর্ডের আপ-ডাউন কীগুলো ব্যবহার করতে পারেন। আর এখান থেকে বের হতে q বাটন চাপ দিলেই হবে।

---

আমার প্রোজেক্ট আমি বাইরে সবার সাথে শেয়ার করতে চাই। এজন্যে আমাদের একটা হোস্ট প্রোভাইডার লাগবে, যে গিট ফ্রেন্ডলি এবং আমাকে গিটের সুবিধাগুলোসহ আমাকে ফ্রাইতে হোস্ট প্রোভাইড করবে। এরকম একটা প্রোভাইডারই হচ্ছে গিটহাব। আরো অনেক আছে, কিন্তু আজকে আমি গিটহাবেই কিভাবে কি করবেন সব দেখাবো। কাছাকাছি ইউজার ইন্টারফেস থাকায় তাই পরে চাইলেও অন্য কোনো প্রোভাইডারের সার্ভিসও ব্যবহার করতে পারবেন।

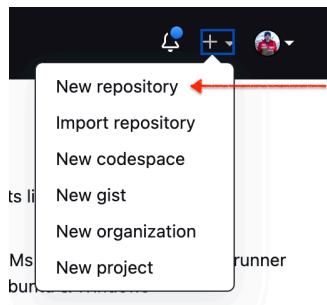
# গিটিহাব

# গিটহাবের সাথে লিঙ্ক

প্রথমে আপনি [গিটহাবে](#)(github.com) একটা অ্যাকাউন্ট ত্রিয়েট করে নিন। [এখান থেকে](#)(github.com/join) অ্যাকাউন্ট ত্রিয়েট করতে হবে। তারপর ইমেইল ভেরিফিকেশনসহ যাবতীয় প্রোফাইলের ইনফরমেশন দিয়ে নিজে নিজে বাকি কাজ করতে পারবেন আশা করি।

এখন আপনার গিটহাবের অ্যাকাউন্ট এ লগিন করলে উপরে ডান পাশে একটা প্লাস চিহ্ন দেখতে পাবেন। সেখানে ক্লিক করলে একটা মেনু ওপেন হবে এখানে **New repository** নামে লেখা দেখতে পাবেন। এখানে ক্লিক করলে আপনাকে নতুন রিপো তৈরী করার পেজে নিয়ে যাবে। কোন কারণে যদি গিটহাবের ইন্টারফেসের ডিজাইন চেঙ্গ হয়ে থাকে তাহলে হয়তো অন্যরকম ইউআই দেখতে

পারেন। বাট আমরা এখন মূলত গিটহাবে New repository ওপেন করব।



এই অপশনটি চুজ করবেন

তারপর এখানে প্রথমে আপনার রিপোজিটরির নাম (যেমন আমি দিলাম learning-git) দিবেন। রিপোজিটরির নাম ইউনিক এবং ইউ-আর-এল ফ্রেন্ডলি হতে হবে অবশ্যই। পরের ডেক্স্ট্রিপশন ফিল্ড অপশনাল, চাইলে কিছু দিতেও পারেন আবার খালিও রাখতে পারেন। এরপরের যে ফিল্ড আসবে সেখানে আপনি কি পাবলিক রিপোজিটরি করবেন নাকি প্রাইভেট করবেন সেটা জিজ্ঞাসা করা হয়েছে। আপনার ক্ষী অ্যাকাউন্ট হয়ে থাকলে শুধুমাত্র পাবলিক রিপোজিটরি করার অ্যাক্সেস পাবেন। তারপর বাকি ফিল্ডগুলা এভাবেই রেখে **Create repository** বাটনে ক্লিক করুন।

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

### Repository template

Start your repository with a template repository's contents.

No template ▾

Owner \*



zonayedpca ▾

Repository name \*



/

Great repository names are short and memorable. Need inspiration? How about [literate-memory](#)?

Description (optional)

Public

Anyone on the internet can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file

This is where you can write a long description for your project. [Learn more](#).

ⓘ You are creating a public repository in your personal account.

[Create repository](#)

এখানে আপনার মনমতো অপশন চুজ করেন

ব্যাস! আপনার গিটহাবে নতুন রিপোজিটরি ক্রিয়েট কমপ্লিট। এখন কিছু ইন্সট্রাকশন দেখবেন যেগুলা আসলেই অনেক দরকারী আপনার পরবর্তি স্টেপগুলোর জন্য। এখানে একদম নতুন গিট রিপোজিটরি বানিয়ে কিভাবে গিটহাবের সাথে কানেক্ট করবেন (প্রথমটা) বা অলরেভিড গিট রিপোজিটরি আছে এমন প্রোজেক্টকে কিভাবে গিটহাবের

সাথে কানেক্ট করবেন (দ্বিতীয়টা) সেই ইন্সট্রাকশন দেওয়া আছে।  
আমাদের যেহেতু গিট রিপোজিটরি অলরেডি আছে, তাই দ্বিতীয়  
ইন্সট্রাকশন অনুযায়ী কাজ করব।

The screenshot shows a GitHub repository page for 'zonayedpca/learning-git'. The repository is public. At the top, there are buttons for Pin, Unwatch, Fork, and Star. Below the header, there are links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

**Quick setup — if you've done this kind of thing before**

Set up in Desktop or HTTPS SSH git@github.com:zonayedpca/learning-git.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

**...or create a new repository on the command line**

```
echo "# learning-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:zonayedpca/learning-git.git
git push -u origin main
```

**...or push an existing repository from the command line**

```
git remote add origin git@github.com:zonayedpca/learning-git.git
git branch -M main
git push -u origin main
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

কি কি করতে পারেন সেগুলো নিয়েও কিছু ইন্সট্রাকশন দেওয়া আছে

# গিটহাবে পুশ

এখন কমান্ড লাইন থেকে আপনার লোকাল ডিরেক্টরিতে থাকা প্রোজেক্টে চলে যান। `git status` দিয়ে শিউর হয়ে নেন সবকিছু কমিট করা আছে কিনা, নাকি কোনো কাজ কমিট করা বাকি আছে। তারপর এটাও চেক করে দেখুন কোন ভাষ্টে আছেন। `main` ভাষ্টে থাকলে আপনি রেডি আপনার প্রোজেক্ট পুশ করার জন্যে।

আগুণ লাগলে করনীয়



১ | `git commit`

২ | `git push`

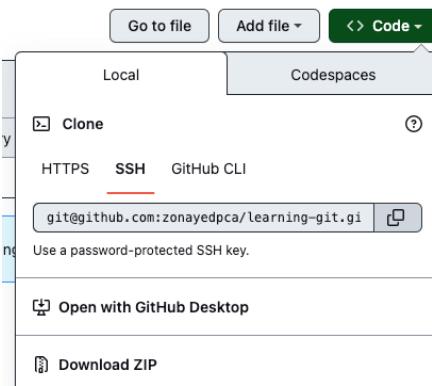
৩। বাইরে চলে যান

কষ্টের কোড পুশ করতে ভুলবেন না কখনই

এখন গিটহাবে দেওয়া ইন্সট্রাকশন অনুযায়ী আমাদের এই প্রোজেক্টে গিটহাবে পুশ করব। তার জন্যে প্রথমে আপনাকে `remote origin` অ্যাড করতে হবে এভাবেঃ

```
> git remote add origin git@github.com:zonayedpca/learning  
-git.git
```

এখানে লিঙ্কটা আপনার প্রোজেক্টের হবে, লিঙ্কটা গিটহাবে ক্লিয়েট করা প্রোজেক্টের লোকেশন থেকে খুঁজে পাবেন। আর এটা SSH লিংক যেহেতু প্রথমে এখানে আমরা SSH ব্যবহার করে প্রোজেক্টে পুল-পুশ করা দেখাবো। লিঙ্কটা আপনি আপনার প্রোজেক্টে গেলে Code বাটনে ক্লিক করে খুব সহজেই পেয়ে যাবেনঃ



এটা আপনার প্রোজেক্টে প্রথমবার অ্যাড করতে হবে, পরের বার পুশ করার সময় লাগবে না। কারণ হচ্ছে আপনার প্রোজেক্ট আপনি কোথায় হোস্ট করতে চাচ্ছেন সেটা আপনার প্রোজেক্টে প্রথমবারই বলে দিতে হবে। পরেরবার থেকে সে মনে রাখবে। তখন শুধু পুশ করলেই হবে। এখন প্রোজেক্ট পুশ করতে চাইলেং:

```
> git push origin master
```

বাট ওয়েট! আপনি যদি পূর্বে গিটহাব সেটআপ করে না থাকেন তাহলে হয়তো এখনি আপনি পুশ করতে পারবেন না। আপনাকে এর আগে আরেকটা ছোট ও গুরুত্বপূর্ণ কাজ করতে হবে।

# SSH কী সেটআপ

আপনি আপনার লোকালি থাকা প্রোজেক্ট যে গিটহাব এ পুশ করবেন, তার আগে আপনাকে অথেন্টিকেটেড পার্সন হতে হবে। ধরুন ফেসবুকে আপনি আপনার অ্যাকাউন্ট থেকে কাউকে ম্যাসেজ পাঠাবেন, কিন্তু একদম নতুন ডিভাইসে এই কাজটা করতে হলে প্রথমে আপনাকে ফেসবুকে ঢুকে ইউজারনেম/ইমেইল, পাসওয়ার্ড দিয়ে লগইন করে ফেসবুককে আপনার পরিচয় দিতে হবে যে আমিই এই আইডির মালিক।

ঠিক তেমনি আপনার লোকাল গিট থেকেও গিটহাবে পুশ করার ক্ষেত্রেও আপনাকে আগে গিটহাবকে প্রমাণ করতে হবে যে আপনার গিটহাব অ্যাকাউন্টে বা আপনিই আপনার প্রোজেক্টে আপলোড অথবা পুশ করবেন। পূর্বে গিটহাবে ইউজারনেম/ইমেইল আর পাসওয়ার্ড দিয়ে

সে কাজটা করা যেতো। বাট এটা একটু কম সিকিউরড হওয়ার কারণে বর্তমানে আপনাকে SSH কী অথবা টোকেন এর সাহায্যে কাজটা করতে হবে।

তবে আপনি আপনার ব্যক্তিগত ম্যাশিনে যেখানে প্রতিনিয়ত এই গিট ও গিটহাব নিয়ে কাজ করবেন সেখানে SSH কী সেটআপ করে নেওয়াটাই সবচেয়ে সেইফ এবং সিকিউরড। আর টোকেন এর ব্যাপারটা সাময়িক সময়ের জন্য অন্য কোন ম্যাশিন থেকে কাজ করতে গেলে তখন করা উচিত।

তবে আমি এখানে আপনাদেরকে প্রথমে কিভাবে SSH কী সেটআপ করবেন সেটা দেখাবো এবং বইয়ের শেষের দিকে কিভাবে টোকেন ব্যবহার করে করবেন সেটাও থাকবে। উইন্ডোজ আর ম্যাক, লিনাক্সের জন্য সেটআপটা একটু আলাদা আলাদা হতে পারে, তবে দিনশেষে ব্যাপারটা সেইমই।

১। ম্যাক বা লিনাক্সের ক্ষেত্রে আপনি আপনার ফেভারিট টার্মিনাল ওপেন করুন। আর উইন্ডোজের ক্ষেত্রে হলে গিট ব্যাসের টার্মিনালটা ওপেন করেন এবং নিচের এই কমান্ডটি লিখুনঃ

```
> ls -al ~/.ssh
```

এখন যদি আপনার সিস্টেমে অলরেডি SSH কী থেকে থাকে(ফাইলের নাম হবে এরকমঃ `id_rsa` এবং `id_rsa.pub`), তাহলে আপনি একদম নিচের ৫ নাম্বার স্টেপে দেখানোর মতো করে পাবলিক কী(যেটা এখানে `id_rsa.pub` ফাইলটি) এর কন্টেন্টগুলো কপি করে নিবেন। আর যদি না থাকে তাহলে পরের স্টেপে ফলো করুন।

২। আপনাকে এখন নতুন SSH কী জেনারেট করতে হবে নিচের কমান্ড অনুযায়ী। তবে নিচে এখানে অন্যান্য সব অপরিবর্তিত রাখলেও ইমেইলটা অবশ্যই আপনার ব্যবহার করা অ্যাকচুয়াল ইমেইল অ্যাড্রেস দিয়ে রিপ্লিস করে নিবেনঃ

```
> ssh-keygen -t rsa -b 4096 -C "zonayedpca@gmail.com"
```

এখানে আমরা `rsa` টাইপের একটা কী যেটার সাইজ হচ্ছে ৪০৯৬ বিটস(গুলো অপরিবর্তিত রাখতে পারেন) জেনারেট করছি আমি আমার ইমেইল দিয়ে(এখানে আপনি আপনার ইমেইল ব্যবহার করবেন অবশ্যই)। এখানে জেনারেট করার সময় আপনি বেশ কিছু অপশন পাবেন, গুলো আপাতত এন্টার দিয়ে দিয়ে কমপ্লিট করে ফেলতে পারেন(`passphrase` ফিল্ডও খালি রাখতে পারেন)। তারপর এখানে এখন একটা পরিলিক কী, আর আরেকটা প্রাইভেট কী জেনারেট হবে।

৩। তারপর আপনাকে SSH এজেন্ট ব্যাকগ্রাউণ্ডে রান করানোর জন্য  
এই কমান্ডটা ব্যবহার করতে হবেঃ

```
> eval "$(ssh-agent -s)"
```

৪। তারপর এই SSH এজেন্টে আপনাকে আপনার নতুন ক্রিয়েট করা  
কিণ্ডলোর মধ্যে প্রাইভেট কী'টাকে(id\_rsa) অ্যাড করতে হবেঃ

```
> ssh-add ~/.ssh/id_rsa
```

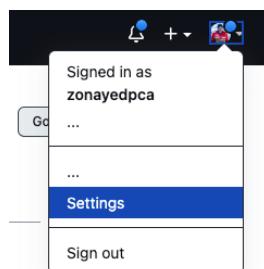
৫। এ বার আ ম রা আ ম দে র পা ব লি ক S S H  
কী'টাকে(id\_rsa.pub) কপি করব। এটা আপনি কোন কোড  
এডিটর, নেটপ্যাড দিয়ে বা যেকোনোভাবেই করতে পারবেন।  
আপনার মেইন উদ্দেশ্য হবে কী'টাকে কপি করা। উইন্ডোজে হলে গিট  
ব্যাশ থেকে এভাবে কপি করতে পারবেনঃ

```
> cat ~/.ssh/id_rsa.pub | clip
```

আর ম্যাক বা লিনাক্সে হলে এভাবে করতে পারেনঃ

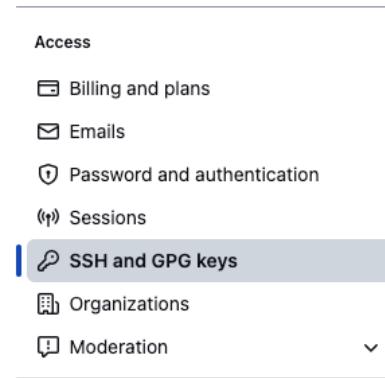
```
> pbcopy < ~/.ssh/id_rsa.pub
```

৬। এবার সর্বশেষগ স্টেপ হচ্ছে আপনার কপি করা এই পাবলিক কীটাকে গিটহাবে নিয়ে রাখতে হবে যেটা দ্বারা গিটহাব আপনাকে আইডেন্টিফাই করতে পারবে। এরজন্য প্রথমে আপনি আপনার গিটহাব অ্যাকাউন্টে লগইন করুন এবং একদম উপরে ডান পাশের কর্ণারে আপনার প্রোফাইল পিকের উপর ক্লিক করে সেটিংস এ যাবেনঃ



Settings এ ক্লিক করবেন

ব্যাস এবার আপনি আপনার বাম পাশে SSH and GPG keys নামে  
একটা মেনু দেখতে পাবেনঃ



SSH and GPG keys এ ক্লিক করবেন

এবার এই মেনুতে গিইয়ে উপরে ডান পাশে New SSH key নামে  
একটা বাটন দেখবেনঃ

New SSH key

at recognize.

New SSH key এ ক্লিক করবেন

এখানে ক্লিক করে এখন আপনি আপনার কপি করা SSH কী(পাবলিক কী) টা এখানে পেস্ট করে দিবেন এবং মনে রাখার সুবিধার্থে যে এই কী'টা আপনার কিসের জন্য ব্যবহার করবেন বা কোন ম্যাশিনের সেটার উপর ভিত্তি করে একটা টাইটেলও লিখে দিবেন। অন্যান্য সব অপরিবর্তিত থাকবে, কী টাইপও অথেন্টিকেশন কী'ই থাকবেং

### SSH keys / Add new

Title

Git and Github Ebook

Key type

Authentication Key

Key

ssh-rsa

AAAAB3NzaC1yc2EAAQABlwAAAQEA879BjGYIPTLJuc9/R5MYIN4yc/YIClccBpSdzgK9Dt0Bkfe3rSz5cPm4wmehdE7GkVEFxjBj2YHqPLuM1x1AUxlebpwlll9f/ajUHQts9eVnVh4NztPyQlSU/l97d8g7fd8g7df7ydf8g7df98g7dfgf8g7fd7g6dfgd78gdf7g687dfg7f18dgdg78fd78g687dfg9ORQa6wvZMVrPECBvwltTY8cPWH3MGZIK/74ehBLSKA4PY3gM4GHI450Nle16yggEa2aTQtWA1rry9YYWEoHS9pJ1dnLaZU3k/8OWaaJrlwSoC5rGiap93iu0H8T6+mEHGRQe84NK1y5IESSWlbn6P636BI3uQ== zonayedpcat@gmail.com



Add SSH key

Title যেকোনোকিছু দিতে পারেন, বাট কী টাইপ এটাই রাখবেন

ব্যাস এবার Add SSH key বাটনে ক্লিক করলেই আপনার সিস্টেমের পাবলিক SSH কী টা গিটহাবে স্টোর হয়ে যাবে। এবার আপনি নিশ্চিন্তে আপনার লোকাল সিস্টেম থেকে আপনার গিটহাবে কোড পুশ করতে পারবেন।

---

তবে গিটহাব SSH কী অ্যাড করার ইউআইটা চেঞ্চও করতে পারে, সেক্ষেত্রে আপনাকে জাস্ট সেটিংস থেকে অথেন্টিকেশনের জন্য SSH কী অ্যাড করার অপশনটা খুঁজতে হবে। আর নতুন আপডেট আসলে আমিও বইয়ে চেষ্টা করব আপডেট করে দিতে। তারপরেও আপনি চাইলে আমাকে বিষয়টি সম্পর্কে অবহিত করতে পারেন।

---

ব্যাস এবার আপনি আপনার প্রোজেক্টে গিয়ে এভাবে পুশ করলেই আপনার প্রোজেক্ট গিটহাবে চলে যাবেং

```
> git push origin main
```

তবে প্রথমবার পুশ করার সময় আপনাকে অথেন্টিকেট করার জন্য বলতে পারে যেখানে আপনি সিম্পলি yes লিখে এন্টার দিবেনঃ

```
> The authenticity...'github.com (...)' can't be established.  
RSA key fingerprint is xx:xx:xx:xx...:xx:xx:xx:xx:xx:xx.  
Are you sure you want to continue connecting (yes/no): yes
```

এখানে আমরা আমাদের প্রোজেক্টের শুধুমাত্র main ভাঞ্চ পাঠিয়েছি।  
অন্য ভাঞ্চগুলো পুশ করতে চাইলে জাস্ট main এর জায়গায় সে  
ভাঞ্চ এর নাম লিখে পুশ করে দিলেই হবে। যেমন আমরা table-  
version টাও যদি পুশ করতে চাইঃ

```
> git push origin table-version
```

# গিটহাব থেকে পুল

এখন ধরলাম আপনার এই প্রোজেক্টে আরো কয়েকজন ডেভেলপার আছে। এদেরকে **Collaborators** ও বলা হয়। আমি পরে এ ব্যাপারে লিখেছি কিভাবে Collaborators অ্যাড করবেন আপনার প্রোজেক্টে। এখন অন্য কোনো Collaborator আপনার প্রোজেক্টে নতুন কোনো কাজ করে সেটা পুশ করেছে গিটহাবে। এখন অটোম্যাটিক্যালিই কিন্তু সেই কাজের আপডেট গিটহাব থেকে আপনার লোকাল ম্যাশিনে চলে আসবে না। সেজন্যে আপনাকে সেটা পুল করতে হবে গিটহাব থেকে এভাবেঃ

```
> git pull origin main
```

এখানে লক্ষণীয় যে পুল করার সময় যদি আপনার এখানে কোন কাজ কমিট করা বাকি থাকে তাহলে আপনি পুল করতে পারবেন না। সেক্ষেত্রে প্রথমে আপনাকে আপনার লোকাল কাজগুলোকে বাদ দিয়ে দিতে হবে অথবা কমিট করে দিতে হবে।

তবে আপনার যদি নিজের কোন নতুন কমিট থাকে যেটা আপনি গিটহাবে পুশ করেননি, তখন এখানে কিন্তু কনফিউট হতে পারে। দুইজন কোলাবোরেটর অথবা আরেক ম্যাশিন(হ্যাএখন কিন্তু চাইলে আপনি একাধিক ম্যাশিন থেকেও সেইম প্রোজেক্টে কাজ করতে পারবেন) থেকে যদি একই ফাইল এডিট করে থাকেন, তাহলে সে কারণে সেখানে গিট যতটুকু সম্ভব অটোম্যাটিক্যালি সেই কাজগুলো মার্জ করার চেষ্টা করবে, ঠিক ব্রাঞ্চ মার্জ করার মতোই।

আর যদি কোনো কনফিউট পায় যেটাতে গিট কনফিউটড, সেক্ষেত্রে গিট সেই লাইনের কোডগুলো স্পেশাল কিছু লেখা দিয়ে হাইলাইট করে দিবে। আপনার তখন ম্যানুয়ালী গিয়ে কোন লাইনটা রাখবেন আর কোনটা বাদ দিবেন সেটা বাছাই করে দিয়ে আবার সেই চেঞ্জগুলো কমিট করে দিতে হবে। আমি এই লেখা সিম্পল রাখতে যাচ্ছি তাই এগুলো নিয়ে বেশি গভীরে ঘাবো না, কিন্তু জেনে রাখা অবশ্যই ভালো।

# নিজের প্রোজেক্টে পুল রিকোয়েস্ট

গিটহাবে সাধারনত মূল কাজ বাই ডিফল্ট মেইন(main) ভাবে হয়ে থাকে। বাই কনভেনশন অন্য ভাষ্ম অন্য কিছু টেস্ট করার উদ্দেশ্যে বানানো হয়ে থাকে। তো আপনি কোনো প্রোজেক্টে কাজ করলে সেই প্রোজেক্টে অনেকজন Collaborators থাকতে পারে। তারমধ্যে হয়তো লিডেও কেউ থাকতে পারেন। এখন লিডের অনুমতি ছাড়া বা সিদ্ধান্ত ছাড়া নতুন কোনো ফিচার হয়তো মাস্টার ভাবে অ্যাড করা নাও যেতে পারে। সেক্ষেত্রে আপনার করা নতুন ফিচার অন্য Collaborators কিভাবে দেখবে?

সিম্পল! আপনি আরেকটা ভাবে কাজ করে সেটা পুশ করে দিবেন গিটহাবে। ধরি আমাদের প্রোজেক্টে আমরা এখন নতুন ভাষ্ম অ্যাড করতে চাচ্ছি আমাদের friend-lists.txt ফাইলটা একটু

মডিফাই করে। প্রথমে নতুন একটা ব্রাঞ্চ বানিয়ে নেই location-version নামেং

```
> git checkout -b location-version
```

এখন friend-lists.txt ফাইলটা একটু মডিফাই করে নিইং

```
=====
|| Dibakar Sutradhar      || Cumilla, Bangladesh
=====

|| S M Shahadat Hossain  || Cumilla, Bangladesh
=====

|| Reduanul Houque Munna || Chattogram, Bangladesh
=====

|| Ar Rolin              || Dhaka, Bangladesh
=====

|| Niraj Paudel          || Pokhara, Nepal
=====

|| Tanvir Faisal Moon    || Cumilla, Bangladesh
=====

|| Sagar Neupane          || Kathmandu, Bangladesh
=====

|| Yadav Lamechane        || Kharar, Punjab, India
=====
```

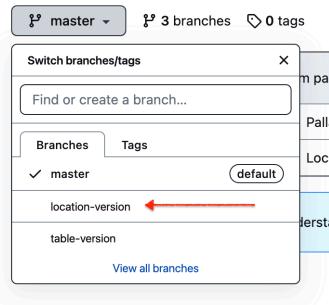
ব্যাস এখন এই মডিফিকেশনটা কমিট করে দেইঃ

```
> git add --all  
> git commit -m "Location added"  
> [location-version 6b1a3da] Location added  
  1 file changed, 8 insertions(+), 8 deletions(-)
```

এখন এই ব্রাঞ্চ গিটহাবে পুশ করে দিবোঃ

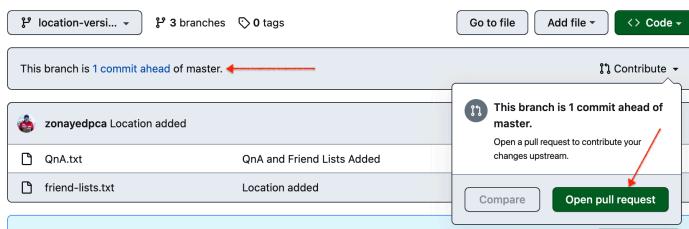
```
> git push origin location-version
```

ব্যাস কোনো এরর না দেখালে আপনার পুশ হয়ে গেছে। এখন গিটহাবে গিয়ে আপনার করা নতুন location-version ব্রাঞ্চে চলে যান। এটা এখানে আপনার প্রোজেক্ট ফাইল লিস্টিং এর বাম পাশে উপরের দিকে পাবেন যেখান থেকে আপনি ব্রাঞ্চ সুইচ করতে পারবেন।



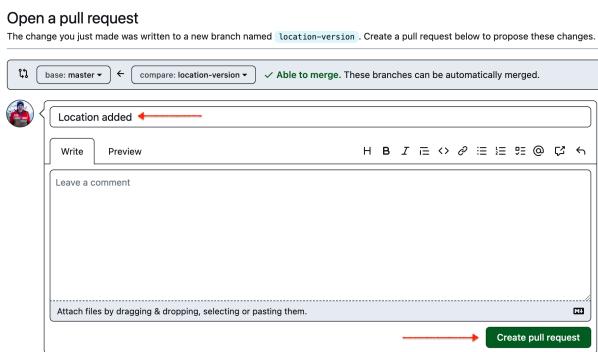
সবগুলো রাখিই দেখাচ্ছে এখানে

এবার location-version এ দেখবেন লেখা রয়েছে **This branch is 1 commit ahead of main**. তো এখন এটা যাতে মাস্টারে অ্যাক্সেপ্ট করা হয় সেজন্যে আপনি এখানে দেখবেন পাশেই **Contribute** নামে একটা অপশন আছে যেটার ভিতর থেকে **Open pull request** নামে একটা বাটন আছে। এখানে ক্লিক করলে পরের পেজে নিয়ে যাবে।



main থেকে ১ কমিট বেশী আছে, পাশেই কন্ট্রিবিউট করার অপশন

এখানে কি কি মডিফাই করা হয়েছে তার বিস্তারিত লিস্ট দেখতে পারবেন, আর পুল রিকোয়েস্ট এর জন্যে কোনো কমেন্ট করতে চাইলে সেটা লেখার সুযোগ পাবেন(অপশনাল)। পরে নিচে **Create pull request** বাটনে ক্লিক করলে ফাইনালি আপনার পুল রিকোয়েস্ট চলে যাবেং।

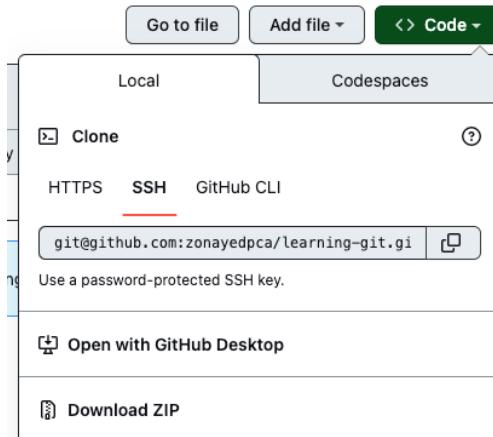


এখানে আমরাই আমাদের নিজেদের প্রোজেক্টে পুল রিকোয়েস্ট করেছি। তবে অনেকজন কোলাবোরেটর থাকলে সেখানেও এভাবে করা যাবে। আর পুল রিকোয়েস্ট নামে ট্যাবে এই রিকোয়েস্ট টা থাকবে। যে প্রোজেক্টে এগুলো মার্জ করার দায়িত্বে থাকবে সে চাইলে সেখানে গিয়ে বিস্তারিত দেখে পুল রিকোয়েস্ট অ্যাক্সেপ্টও করতে পারবে আবার চাইলে বাদও দিয়ে দিতে পারবে। তবে এখানে যেহেতু সবাই কোলাবোরেটর তাই যেকেউ চাইলেই এটা মার্জ করতে পারবে, কিন্তু প্রোজেক্টে সাধারণত ঠিকমতে ফ্লো বজায় রাখতে ঘার যেটা দায়িত্ব সে সেটা নিয়েই কাজ করে।

# গিটহাব থেকে প্রোজেক্ট ক্লোন

এখন গিটহাবে থাকা কোনো প্রোজেক্ট আমাদের লোকাল ম্যাশিনে আনতে চাইলে সেটাকে ক্লোন করতে হয়। অর্থাৎ গিটহাবে থাকা কোনো প্রজেক্টের একটা ক্লোন কপি আপনার লোকাল ম্যাশিনে নামিয়ে আনতে চান। যে প্রোজেক্ট আপনার কাছে নাই, একদম নতুন। আপনি চাইলে গিটহাবের যে কোনো পাবলিক প্রোজেক্টই ক্লোন করতে পারবেন। সেক্ষেত্রে জাস্ট কমান্ড লাইন ওপেন করে ক্লোনের কমান্ড দিলেই পুরো প্রোজেক্ট আপনার লোকাল ম্যাশিনে চলে আসবে।

এই জন্যে প্রথমে আপনার ক্লোন এর লিঙ্ক খুজতে হবে। ক্লোন করার লিঙ্ক আপনি গিটহাবের যেকোনো প্রোজেক্টের পাতায় গিয়ে প্রোজেক্ট ফাইল লিস্টিং এর ডান দিকের উপরের কোনায় পাবেন <> **Code** নামের বাটনে।



আমরা যেহেতু SSH ব্যবহার করছি তাই এটা সিলেক্ট করলাম, বাট HTTPS ব্যবহার করলে এখানে HTTPS ই সিলেক্ট করতে হবে।

এখানে অনেকরকম অপশনই আছে, এখান থেকে জিপ ফাইলও নামাতে পারবেন। তাছাড়া আপনি যদি HTTPS ব্যবহার করেন(এখনই না করে থাকলে এই বইয়ের শেষের দিকে আমরা সেটা দেখবো) তাহলে HTTPS ট্যাব থেকে ক্লোন এর ইউআরএলটা নিতে হবে। বাট আমরা আপাতত এগুলো কোনটাই করব না, আমরা SSH ব্যবহার করে কমান্ড লাইনের সাহায্যে ক্লোন করব এভাবেঃ

```
> git clone <GitHub Repo URL> <Local Directory  
Name(optional)>
```

এভাবে প্রথমে clone তারপরে গিটহাবের রিপোজিটরির লিঙ্ক। তারপরে আপনার লোকাল ম্যাশিনে প্রোজেক্টটা কোন ডিরেষ্টরির ভিতরে রাখতে চাচ্ছেন সেটার নামও দিতে পারবেন। লোকাল ডিরেষ্টরির নাম দেওয়াটা অপশনাল, না দিলে বাই ডিফল্ট রিপোজিটরির যে নাম সে নামের ডিরেষ্টরিতেই ক্লোন হবে। ধরি, আমরা এই প্রোজেক্টটা ক্লোন করব। তাই কমান্ড লাইন ওপেন করে কমান্ড লিখবোঃ

```
> git clone git@github.com:zonayedpca/learning-git.git  
learning-git-with-zonayed
```

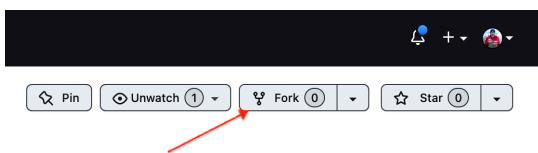
এখানে আমার লেখা ফলো করে থাকলে আপনিও হয়ত সেইম নামের রিপোজিটরি নিয়ে কাজ করছেন। তাই আমরা অপশনাল আর্গুমেন্টটাও দিলাম আমাদের মন মতো learning-git-with-zonayed নামক ডিরেষ্টরিতে প্রোজেক্ট ক্লোন করার জন্যে।

এখন এন্টার দিলে প্রোজেক্ট আস্তে আস্তে ক্লোন হয়ে যাবে আপনার লোকাল ম্যাশিনে। ধরি, এটা আমি আমার ডেস্কটপে ক্লোন করেছি। এখন ডেস্কটপে দেখবেন ক্লোন করার পর learning-git-with-zonayed নামে একটা প্রোজেক্ট চলে আসছে। আর এটাই আপনার ক্লোন করা রিপোজিটরি।

এখন এটার ভিতরে গিয়ে আবার চাইলে কাজ করে গিট  
ব্যবহার করতে পারবেন, লোকালি প্রোজেক্ট মডিফাই করতে পারবেন,  
সবই করতে পারবেন। আর এই প্রোজেক্টার remote আপনি যেখান  
থেকে ক্লোন করেছেন সেটাই সেট করা থাকবে অটোম্যাটিক্যালি।  
আপনি এই প্রোজেক্টের Collaborator না হয়ে থাকলে এই রিমোটে  
পুশ করতে পারবেন না, আর Collaborator হয়ে থাকলে এই  
প্রোজেক্টেই পুশ করতে পারবেন আপনার করা নতুন কোনো কমিট।

# অন্য প্রোজেক্টে পুল রিকোয়েস্ট

এখন ধরলাম আপনি একটা প্রোজেক্টে কন্ট্রিবিউট করতে চাচ্ছেন। বা এখানে আপনি আমার ডেমো প্রোজেক্টে কন্ট্রিবিউট করতে চাচ্ছেন। তো সেক্ষেত্রে প্রথমে আমার গিটহাবের প্রোজেক্টে গিয়ে সেটা **fork** করতে হবে। এই **fork** বাটন গিটহাবের কাঞ্জিক্ষত প্রোজেক্টের পেজে একদম উপরে ভান কোণায় পাবেন।



সরাসরি Fork বাটনে ক্লিক করবেন

ফর্ক এ ক্লিক করলে আপনি কোথায়(Owner) সেটা ফর্ক করতে চাচ্ছেন সে অপশন আসবে। আপনার গিটহাবে কোনো অর্গানাইজেশন

না থেকে থাকলে আপনার অ্যাকাউন্টই দেখাবে। আপনার অ্যাকাউন্টের উপরে ক্লিক করলে এটা ফর্ক হয়ে যাবে আপনার অ্যাকাউন্টে।

### Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (\*).

Owner \*



Repository name \*

/ learning-git

learning-git is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

Copy the `master` branch only

Contribute back to [zonayedpca/learning-git](#) by adding your own branch. [Learn more.](#)

① You are creating a fork in the WithZonayed organization.

**Create fork**

এখানেও কিছু তথ্য দিতে পারেন, তারপর Create fork এ ক্লিক করুন

ফর্ক অনেকটা ক্লোনের মতই। ক্লোন করলে যেমন আপনার লোকাল ম্যাশিনে একটা কপি ডাউনলোড হয়। কিন্তু ফর্ক করলে লোকাল ম্যাশিনে কপি ডাউনলোড না হয়ে আপনার গিটহাব অ্যাকাউন্টে একটা কপি তৈরি হবে। সেই কপিটাকে আপনি আপনার রিপোজিটরি হিসাবে ব্যবহার করতে পারবেন। সেই ফর্ক করা রিপোজিটরিকে ক্লোন করে লোকাল ম্যাশিনেও নামিয়ে আনতে পারবেন।

এখন ফর্ক হচ্ছে এই প্রোজেক্টেরই বর্তমান ভার্শনটা আপনার অ্যাকাউন্টে কপি করে ফেলা, আর কোনোভাবে এটা মেইন প্রোজেক্টের সাথে লিঙ্কড থাকে যাতে পরে আরো কিছু জিনিস করতে পারেন। এখন এটা ক্লোন করে আপনি মডিফাই করে আবার আপনার ভার্শনেও পুশ করতে পারবেন। ফর্ক করার পর প্রজেক্টটা ক্লোন করে ফেলুন এভাবেঃ

```
> git clone git@github.com:WithZonayed/learning-git.git
```

এখানে আপনার ফর্ক করা প্রোজেক্টের ক্লোন লিঙ্কটা হবে। এবার প্রোজেক্টের ডিরেস্টরিতে গেলে দেখবেন আমি একটা ফাইল রেখেছি QnA.txt নামে। এখান আপনি আপনার নাম এবং সাথে আপনার তৈরী করা প্রথম গিটহাবের রিপোজিটরির লিঙ্ক টা জাস্ট পেস্ট করবেন। এখানে এমন অলরেডি কয়েকটা লিঙ্ক দেখতে পাবেনঃ

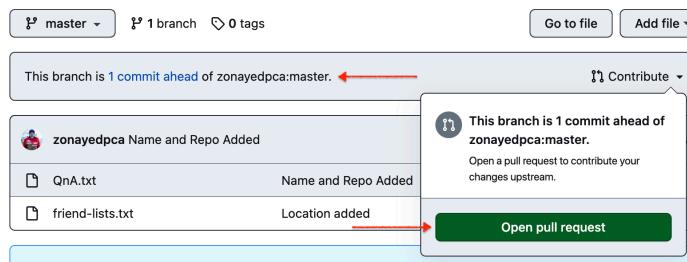
- [Zonayed Ahmed] <https://github.com/zonayedpca/learning-git>

আগের থাকা লিঙ্কগুলো মুছবেন না। ওগুলোর নিচে আপনারটা পেস্ট করবেন। তারপর এটা কমিট করে পুশ করে দেন আপনার প্রোজেক্টেঃ

```
> git add --all  
> git commit -m "Name and Repo Added"  
> git push origin master
```

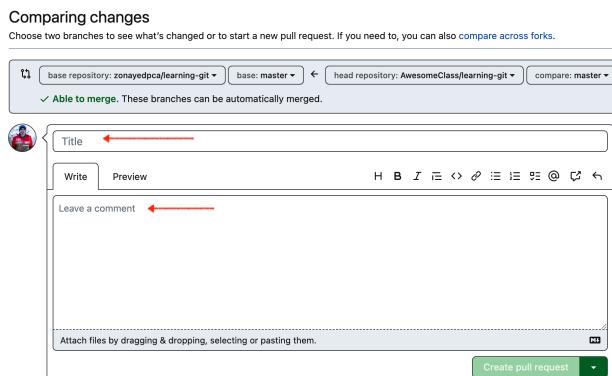
ব্যাস এবার আপনার পুশ কমিট হয়ে গেলে আপনার গিটহাব প্রোফাইল থেকে আপনার ফর্ক করা প্রোজেক্টে গিয়ে দেখবেন নতুন এই কমিট পুশ হয়েছে।

এখন গিটহাবে আপনার ফর্ক করা প্রোজেক্টে গিয়ে দেখবেন এখানে মেইন প্রোজেক্ট থেকে একটা কমিট এগিয়ে আছে দেখাচ্ছে। আর এর ঠিক ডান পাশেই আছে **Contribute** বাটনটি যেখানে গেলে আপনি **Open pull request** নামের বাটনটি দেখতে পাবেন। এই বাটনটি ব্যবহার করেই আপনি মেইন প্রোজেক্টে আপনার করা চেঞ্জটার জন্য পুল রিকোয়েস্ট তৈরি করতে পারবেন।



সেইম পূর্বে দেখানোর মতোই

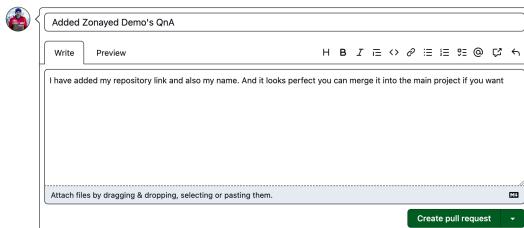
এটাতে ক্লিক করলে পরের পেজে আপনাকে মেইন প্রোজেক্ট এর সাথে আপনার ফর্ক করা প্রোজেক্ট এর চেঞ্জেসগুলো দেখাবে। এখানে যদিও লেখা দেখছেন Able to merge, তবে কোন কারণে আপনি আপনার ফর্ক করা প্রোজেক্টে কাজ করতে করতে মেইন প্রোজেক্টে যদি কোন অপড়েট আসে তাহলে এটা Can't automatically merge লেখা উঠতে পারে। বাট যেটাই হউক আপনি এখানে থেকে **Create pull request** নামে বাটন পাবেন। এখানে আপনার চেঞ্জের একটা টাইটেল আর চাইলে আরো কিছু বিস্তারিত তথ্য কমেন্ট হিসেবেও দিতে পারেনঃ



টাইটেল লাগবেই, কমেন্ট অপশনাল

টাইটেল দেওয়ার পরেই দেখবেন **Create pull request** বাটনটা এনাবল হয়েছে, অর্থাৎ আপনি এখন চাইলেই পুল রিকোয়েস্ট তৈরি

করতে পারবেন। অনেকসময় টাইটেল ফিল্ডে আপনার করা চেঞ্জ বা চেঞ্জেসের কমিট ম্যাসেজ বাই ডিফল্ট পূরণ করা থাকতে পারে।



আমি কিছু তথ্য দিলাম

আপনার চেঞ্জগুলো আর মেইন প্রোজেক্টের মধ্যকার বিস্তারিত ডিফারেন্স বা কম্পারিজন দেখতে পাবেন একটু নিচে স্ক্রল করলে। এটাকে আপনাকে একটা আইডিয়া দিবে যদি আপনার পুল রিকোয়েস্ট মেইন প্রোজেক্টের সাথে মার্জ হয় তাহলে সেটা কেমন হবে।

প্রোজেক্টওয়াইস পুল রিকোয়েস্টের টাইটেল, কমেন্ট কিভাবে লিখতে হবে সেটারও নির্দিষ্ট কোন নিয়ম বা ফরম্যাট থাকতে পারে, থাকলে সেটা প্রোজেক্টের সাথে দেওয়াই থাকবে। তখন আপনাকে তাঁদের দেওয়া নিয়মমতো সবকিছু করতে হবে। বাট আমাদের এখানে সিম্পল রাখার সুবিধার্থে আপনি যেকোনো(অর্থবোধক) কিছুই ব্যবহার করতে পারবেন।

ব্যাস এখন আপনি **Create pull request** বাটনে ক্লিক করে পুল রিকোয়েস্ট পাঠিয়ে দিতে পারবেন। এখন অরিজিনাল কন্ট্রিবিউটর আপনার চেঙ্গ মার্জ করলে আপনিও সে প্রোজেক্টের কন্ট্রিবিউটর লিস্টে অ্যাড হয়ে যাবেন। আমার এই প্রোজেক্টে আমার কথামতো নাম আর রিপোর ইউআরএল দিয়ে পুল রিকোয়েস্ট করলে আমি অ্যাঙ্কেপ্ট করে নিবো। তাহলে আপনিও আমার এই প্রোজেক্টের কন্ট্রিবিউটর লিস্টে অ্যাড হয়ে যাবেন।

# প্রোজেক্ট কন্ট্রিবিউট

# প্রোজেক্ট খোঁজা

কোন প্রোজেক্টে কন্ট্রিবিউট করতে হলে অবশ্যই আপনাকে প্রথমে কোন প্রোজেক্টে কন্ট্রিবিউট করা যায় সে প্রোজেক্টটা খুঁজে বের করতে হবে প্রথমে। যদি আপনি কোন অফিসে বা টিমে কাজ করে থাকেন তাহলে তো সেখানকার প্রোজেক্টেই কন্ট্রিবিউট করা শুরু করতে পারবেন। কিন্তু এর বাইরে আপনি অনেক ওপেন সোর্স, পাবলিককি এভেইলেবল প্রোজেক্টেই কন্ট্রিবিউট করতে পারবেন।

এরকম ছোটো ছোটো কন্ট্রিবিউশ দিয়েই অসংখ্য প্রোজেক্ট বড় বড় প্রোজেক্ট হয়ে গিয়েছে, একটা ম্যাচিউরড অবস্থানে চলে আসছে, বছরের পর বছর চলমান রয়েছে। এরকম অসংখ্য ওপেন-সোর্স প্রোজেক্ট পাবেন যেগুলোতে হাজার হাজার মানুষ অল্প অল্প করে কন্ট্রিবিউট করে প্রোজেক্টকে একটা ভালো অবস্থানে নিয়ে আসছে।

এসব প্রোজেক্টে সবাই কন্ট্রিবিউট করে প্রধানত অনেকগুলো কারণে, এরমধ্যে সবচেয়ে বড় কারণ হলো আমাদের নিজের জন্যেই আমরা কন্ট্রিবিউট করি। ধরেন আমাদের একটা সিকিউরড ফাইল ডাউনলোড করার প্লাগিন দরকার আমাদের ওয়েবসাইটে। এখন আমার টেকনিক্যাল স্কিল থাকলে আমি সময় খরচ করে সেটা একদম স্ক্র্যাচ থেকে কোড করে, টেস্ট করে, সিকিউরিটিসহ ঘাবতীয় বিষয়াদি চেক করে তারপর সেটা আমার ওয়েবসাইটে ব্যবহার করতে পারি।

কিন্তু এমন যদি হতো যে এমন একটা প্লাগিন অলরেডি তৈরি করে অসংখ্য মানুষ, টেস্ট করে, ঘাবতীয় সবকিছু পার্ফেক্ট করে রেডি করে রেখে দিয়েছে আপনার(সবার) জন্য? জি ওপেন সোর্স প্রোজেক্টগুলোর কাজই আসলে এটা। এমন অসংখ্য প্রোজেক্ট পাবেন যেগুলো আমাদের নিত্যদিনের জীবন অনেক সহজ করে দিবে। আমাদের অলমোস্ট কোনকিছু ভাবতেই হবে না, কারণ এসব প্রোজেক্ট ওপেন-সোর্স হওয়াতে, সবার কন্ট্রিবিউশন দ্বারা অলরেডি পার্ফেক্ট, টেস্টেড হয়ে বসে আছে।

তারপরেও এসব প্রোজেক্ট নিয়ে কাজ করতে গেলে দেখবেন যে অনেকসময় কোথাও কোথাও টুকটাক বাগ, বানান ভুল বা একটা দরকারি ফাংশনালিটির অভাব রয়ে গেছে। ঠিক তখনই কিন্তু আপনি জাম্প করতে পারেন সে প্রোজেক্টে কন্ট্রিবিউট করার জন্য। আপনি

যেমন জিনিসটা থেকে উপকার পেলেন, ঠিক সেভাবে কিছু ব্যাক করে দেওয়ারও ভালো একটা উপায় হচ্ছে এটা।

---

আমার এই বইয়ে বানান ভুল থেকে শুরু করে অন্যকোন সমস্যা যদি খুঁজে পান তা হলে আমাকে কিন্তু জানাতে পারেন [contact@zonayed.me](mailto:contact@zonayed.me) এই ইমেইলে। এটাও একরকম কন্ট্রিবিউশন হবে আপনার আমার এই বইয়ে।

---

আর হ্যাঁ এই গিট ও গিটহাবের পাওয়ার ব্যবহার করেই কিন্তু এরকম অসংখ্য মানুষ সেইম প্রোজেক্টেই কন্ট্রিবিউট করতে পারে খুব সহজেই। আজকে আমরা ঠিক তেমন একটা প্রোজেক্টে কন্ট্রিবিউট করা দেখাবো।

আমরা যে প্রোজেক্টে কন্ট্রিবিউট করব সেটা আমাদেরই একটা ইনিশিয়েটিভ প্রোজেক্ট। বাট কাজটা পুরোটা আমি করিনি, বরং আমাদের দেশের কমিউনিটির ভাই-ব্রাদারদের কন্ট্রিবিউশন দ্বারাই এটা এতদুর চলে আসছে। আজকে এই বই পড়ার পর আপনিও অংশ নিতে পারবেন আমাদের এই প্রোজেক্টে কন্ট্রিবিউটর হিসেবে। জাস্ট এই প্রোজেক্ট কেনো, আশা করি যেকোনো প্রোজেক্টেই ভবিষ্যতে কন্ট্রিবিউট করার ক্ষমতা রাখবেন এটা পড়ার পর।

প্রোজেক্টটির নাম হচ্ছে ডেভসংকেত([github.com/devsonket/devsonket.github.io](https://github.com/devsonket/devsonket.github.io)) , ডেভসংকেত এর কাজ হচ্ছে বিভিন্ন বিষয়, যেমন কোন স্পেসেফিক সফটওয়্যার এর শর্টকাট কী, কোন প্রোগ্রামিং ল্যাংগুয়েজের সিনট্যাক্স, ফ্রেমওয়ার্ক বা লাইব্রেরী বিভিন্ন কাজের সিনট্যাক্স, কোড ইত্যাদি ইত্যাদির উপর চিটশিট তৈরি করা। এসব চিটশিট দেখে বা জাস্ট চোখ বুলিয়েই যাতে পরে আপনার সিনট্যাক্স বা স্পেসেফিক কমান্ড/শর্টকাট কি এর কথা মনে পড়ে যায় সেটাই হচ্ছে এটার মূল উদ্দেশ্য। বাকিটা প্রোজেক্ট এর মূল ওয়েবসাইট([devsonket.com](https://devsonket.com)) দেখলেই আশা করি ধারনা করতে পারবেন।

ডেভসংকেত প্রোজেক্টে আমার Astro JS রিলেটেড চিটশিটটা([devsonket.com/astro-js](https://devsonket.com/astro-js)) দেখছিলাম, হঠাৎ খেয়াল করলাম এখানে কিছু বানান ভুল আছে। Framework এই শব্দটা সচরাচর বাংলায় আমরা “ফ্রেমওয়ার্ক” হিসেবেই লিখি। কিন্তু এখানে সব জায়গায় দেখা যাচ্ছে লেখা আছে “ফ্রেমওয়ার্ক”:

## Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক

Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক এর চিটশিট। এ প্রজেক্টটি স্ট্যাটিক এইচটিএমএল এবং ক্যাম্পান্স প্রিয় ফ্রেমওয়ার্ক

কন্ট্রিভিউটর

---

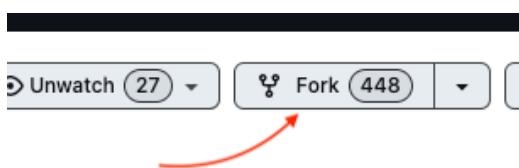
একটা বিষয় লক্ষণীয় যে আমি যেহেতু এখানে এটা ফিল্ড করে কন্ট্রিভিউশন দেখাবো তাই আর উক্ত লিংকে গিয়ে এটা আর দেখতে পাবেন না। তাই আমি এই ওয়েবসাইটের একটা ভার্শন এই astro-typo-devsonket.netlify.app/astro-js লিঙ্কে তৈরি করে রেখেছি। আর এটার গিটহাব রিপোজিটরি পাবেন [github.com/WithZonayed/mistake-devsonket.github.io](https://github.com/WithZonayed/mistake-devsonket.github.io) এই লিংকে। ভুল ভার্শনটা দেখতে চাইলে এই লিংকগুলো ব্যবহার করতে পারেন। কন্ট্রিভিউশন প্র্যাকটিস করতে চালেও এই রিপোতে পুল রিকোয়েস্ট দিতে পারেন, কিন্তু অ্যাক্সেপ্ট করা না হলেও আপনি সফলভাবে করতে পেরেছেন কিনা সেটা জানানো হবে। আপনার প্র্যাকটিস করা হবে!

---

ডেভসংকেত এর উপরে ডান পাশে **এডিট করুন** নামে একটা বাটন রয়েছে, যেখানে ক্লিক করলে আপনাকে এই চিটশিটটার সোর্স কোড যেখানে আছে ঠিক সেখানে নিয়ে যাবে। সব প্রোজেক্টে এরকমটা হবে না, তাই আপনাকে ম্যানুয়ালিও আরো নানানভাবে খুঁজে বের করতে হতে পারে।

# প্রোজেক্ট ফর্ক

কোন প্রোজেক্টে কন্ট্রিবিউট করার অনেক উপায় থাকতে পারে, আজকাল গিটহাব বা এরকম অন্যান্য সার্ভিসগুলো থেকে আপনি সরাসরি ইউজার ইন্টারফেস ব্যবহার করেও কন্ট্রিবিউট করে ফেলতে পারবেন। কিন্তু এখানে আমরা গিট এর কমান্ড লাইন ব্যবহার করে কন্ট্রিবিউট করা দেখাবো। সেজন্যে আমাদেরকে প্রথমেই প্রোজেক্টে ফর্ক(Fork) করতে হবে। প্রোজেক্টের লিংক [github.com/devsonket/devsonket.github.io](https://github.com/devsonket/devsonket.github.io) এ গেলে উপরে ডান পাশে দেখবেন যে একটা ফর্ক বাটন আছে:



ফর্কের কাজ হচ্ছে এই প্রোজেক্ট এর রিপোজিটরির একটা কপি তৈরি করা আপনার নিজের কাছে। তো ফর্ক বাটনে ক্লিক করলে কোথায়(Owner) ফর্ক করবেন, কি নামে করবেন এরকম কিছু তথ্য চাইতে পারে। চাইলে কিছু তথ্য পরিবর্তন রাখতে পারেন, অথবা এভাবেই **Create Fork** বাটনে ক্লিক করতে পারেনঃ

### Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Owner \*                          Repository name \*

 zonayedpcadotcom / devsonket.github.io 

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

  সম্পূর্ণ বাংলায় ডেভেলপার চিটামিট  

Copy the `develop` branch only

Contribute back to devsonket/devsonket.github.io by adding your own branch. [Learn more.](#)

① You are creating a fork in your personal account.

**Create fork**

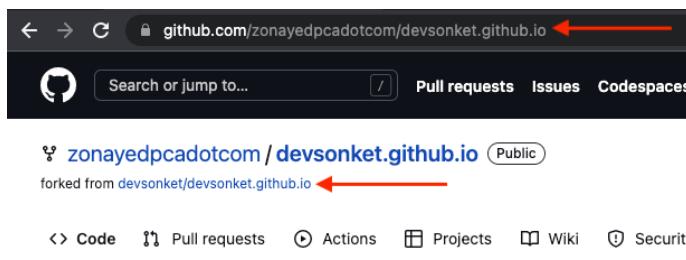


এখানেও কিছু তথ্য দিতে পারেন, তারপর Create fork এ ক্লিক করুন

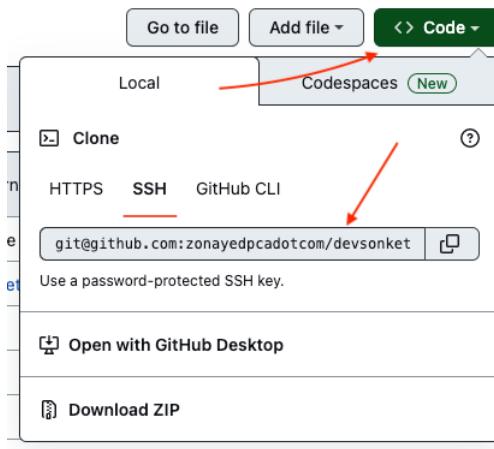
এখানে ক্লিক করার পর আপনার গিটহাবে এই প্রোজেক্টের একটা কপি তৈরি হয়ে যাবে। এখন আপনি চাইলে এটাতে কন্ট্রিবিউট করা শুরু করতে পারেন।

# কন্ট্রিভিউট

প্রোজেক্ট ফর্ক করা হয়ে গেলে এবার আমরা কন্ট্রিভিউট করা শুরু করব। এখন আপনি আপনার ফর্ক করা প্রোজেক্টিকে গিট এর সাহায্যে ক্লোন করে আপনার লোকালে নিয়ে যাবেন। প্রথমেই শিউর হয়ে নিন যে আপনি আপনার ফর্ক করা রিপোজিটরিতেই আছেন, উপরে বাম পাশে খেয়াল করলে, বা গিটহাবের ইউআরএলটা খেয়াল করলেই বুঝতে পারবেনঃ



এবার আমরা প্রোজেক্টটার ক্লোন করে লোকালে নিয়ে আসবো।  
প্রথমেই ক্লোন করার লিংকটা কপি করুন এখান থেকেঃ



এখানে লক্ষণীয় হচ্ছে আপনি যদি SSH পূর্বে সেটআপ করে না থাকেন, HTTPS ব্যবহার করে থাকেন, তাহলে আপনাকে HTTPS ট্যাব থেকে ক্লোনের লিংকটা কপি করতে হবে। আপনার এই গিটহাব অ্যাকাউন্টের সাথে SSH সেটআপ করা থাকলে আপনি SSH লিংকটা ব্যবহার করতে পারবেন।

এবার আপনি আপনার লোকালে যেখানে প্রোজেক্টটি ক্লোন করতে

চান সেখানে কমান্ড লাইন খুলে প্রোজেক্টটি ক্লোন করুনঃ

```
> git clone git@github.com:zonayedpcadotcom/  
devsonket.github.io.gitdevsonket.github.io.git
```

এটা এখন আপনার লোকালে আপনার রিপোজিটরির নাম অনুযায়ী ডিরেক্টরিতে প্রোজেক্টটি ক্লোন করবে, এখানে যেটা হচ্ছে devsonket.github.io। তবে আপনি এ নাম ছাড়া যদি আপনার মনমতো নাম(ধরেন এখানে শুধু devsonket) দিতে চান তাহলে এভাবে কমান্ড দিতে হবেঃ

```
> git clone git@github.com:zonayedpcadotcom/  
devsonket.github.io.git devsonket
```

এ বা র আ ম রা প্রো জে ক্ষে র ডি রে ক্ষে রি র ভি ত রে c d <YOUR\_DIR\_NAME> গিয়ে নতুন একটা ব্রাঞ্চ astro-typo-fix নামে তৈরি করে সেটাতে চেকআউট করব। এখানে আপনি আপনার মনমতো ব্রাঞ্চের নাম দিতে পারেন কোন সমস্যা নেইঃ

```
> git checkout -b astro-typo-fix
```

এবার জাস্ট শিউর হওয়ার জন্য স্ট্যাটাস চেক করে দেখবোঃ

```
> git status
```

এখন On branch astro-typo-fix বা আপনার দেওয়া খানের নাম এখানে দেখতে পাবেন। এরমানে আপনি বর্তমানে আপনার কাঞ্জিক্ষণ খানেই আছেনঃ

```
> On branch astro-typo-fix  
nothing to commit, working tree clean
```

এখন আমরা আমাদের আসল কন্ট্রিভিউশনটা করব, প্রথমেই ভুলটা কোন ফাইলে আছে সেটা খুঁজে বের করব। এখানে ডেভসংকেত এর এডিট করুন বাটন ব্যবহার করেই আমরা খুব সহজেই কোন ফাইলে কন্টেন্টগুলো আছে সেগুলো দেখতে পাচ্ছি। আপনার প্রোজেক্টের ক্ষেত্রে আপনাকে একটু এরকম অপশন না পেলে একটু খোঁজাখুঁজিও করতে হতে পারে। আপনার কোড এডিটরের সার্চ অপশন বা নানানরকমভাবে সেটা খুঁজে পেতে পারেন। তো আমাদের এই ভুলটা হচ্ছে /data/astrojs.json এই ফাইলের ভিতরে। তো আমরা কোড এডিটর দিয়ে সেখানে নেভিগেট করে আমাদের ভুল

"ফ্রেমওয়ার্ক" বানানটা ফিঙ্গ করে সঠিক বানান "ফ্রেমওয়ার্ক"  
বসাবো। পূর্বে এরকম ছিলোঃ

...

```
"id": "astro-js",
"title": "Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক",
"slug": "astro-js",
"description": "Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক এর চিটশিট। যা একটি স্ট্যাটিক  
এইচটিএমএল এবং কম জাভাস্ক্রিপ্ট প্রিয় ফ্রেমওয়ার্ক",
```

...

বানান ফিঙ্গ করার পরঃ

...

```
"id": "astro-js",
"title": "Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক",
"slug": "astro-js",
"description": "Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক এর চিটশিট। যা একটি স্ট্যাটিক  
এইচটিএমএল এবং কম জাভাস্ক্রিপ্ট প্রিয় ফ্রেমওয়ার্ক",
```

...

ফিল্ম করার পড় ফাইল সেইভ করে কমান্ড লাইনে git status  
লিখুনঃ

```
> git status
```

দেখবেন যে আপনার একটা ফাইল মডিফাই করা হয়েছে এমন কিছু  
দেখাহেঃ

```
On branch astro-typo-fix
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:   data/astrojs.json
no changes ... to commit(use "git add" and/or "git commit -a")
```

আরো যদি দেখতে চান তাহলে git diff কমান্ডটাও ব্যবহার করতে  
পারেনঃ

```
> git diff
```

এখানে আরো ডিটেইল্ড পরিবর্তনগুলো দেখতে পাবেন। এবার আমরা এই চেঞ্জেসগুলো অ্যাড করে ফাইনাল কমিট করে দিবোঃ

```
> git add --all  
> git commit -m "astrojs typo fix"
```

ব্যাস হয়ে গেলো আপনার কমিট করা। এখন আপনি এই পরিবর্তনগুলো আপনার গিটহাবে কঙ্কিত ভাষ্টে(এখানে astro-typo-fix) পুশ করবেন এভাবেঃ

```
> git push origin astro-typo-fix
```

ব্যাস আপনার ফিল্টা আপনার ফর্ক করা গিটহাবের রিপোজিটরির কঙ্কিত ভাষ্টে পুশ হয়ে গেলো। আপনি গিটহাবে গিয়েও উক্ত ভাষ্টে(astro-typo-fix) গিয়ে আপনার পরিবর্তনগুলো দেখতে পাবেন। এখন আমরা আমাদের এই ফিল্ট মূল প্রোজেক্টে কন্ট্রিবিউট করব।

এবার আপনার ফর্ক করা গিটহাব রিপোজিটরিতে গেলে দেখবেন উপরে কন্ট্রিবিউট করার জন্য আমন্ত্রণ জানান হচ্ছেঃ

astro-typo-fix had recent pushes less than a minute ago

Compare & pull request

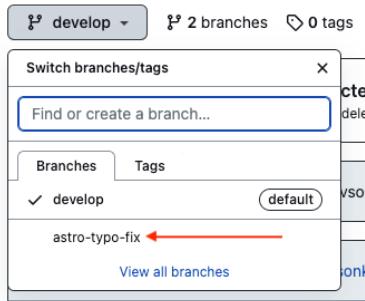
Go to file Add file Code

This branch is 1 commit ahead of devsonket:develop.

Contribute Sync fork

zonayedpcadotcom astrojs typo fix 21e2e12 6 minutes ago 2,618 commits

যদি এরকম কোন নোটিশ নাও দেখতে পান তাহলেও সমস্যা নেই,  
আপনি জাস্ট আপনার ব্রাঞ্চে সুইচ করবেন এখানে থেকেঃ



সুইচ করার পর দেখবেন উপরের দিকে এরকম আপনার ব্রাঞ্চ একটা  
কমিট বা আপনি যে কয়টা কমিট করেছিলেন তত কমিট এগিয়ে আছে  
মূল ব্রাঞ্চ থেকে এমন একটা লেখা এবং এর পাশেই কন্ট্রিবিউট করার  
বাটনটা দেখা যাচ্ছেঃ

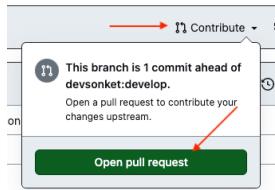
astro-typo-fix 2 branches 0 tags Go to file Add file Code

This branch is 1 commit ahead of devsonket:develop.

Contribute Sync fork

zonayedpcadotcom astrojs typo fix 21e2e12 9 minutes ago 2,618 commits

এবার এই কন্ট্রিবিউট বাটনটাই ব্যবহার করে আমরা মূল প্রোজেক্টে  
কন্ট্রিবিউট করব়ং



**Open pull request** এ ক্লিক করার পর এবার আমাদের পুল  
রিকোয়েস্ট তৈরি করার জন্য বিস্তারিত একটা পেজে নিয়ে যাবে।  
এখানে আপনি আপনার কন্ট্রিবিউশন অনুযায়ী টাইটেল(পুল  
রিকোয়েস্টে একটা মাত্র কমিট থাকলে বাই ডিফল্ট সেই কমিট  
ম্যাসেজই টাইটেল ফিল্ডে দেখাবে) দিবেন আর সাথে কিছু  
বর্ণনা(Description):

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base repository: devsonket/devsonket.github.io base: develop head repository: zonayedpcadcom/devsonk... compare: ast

Able to merge. These branches can be automatically merged.

astros typo fix

Write Preview

এখানে Framework বানানটাকে হালোয় “ফ্রেডওয়ার্ক” না লিখে “ফ্রেডওয়ার্ক” লেখা হয়েছে। এই পুল রিকোয়েস্ট এই বাসনটাইকেই ফিঙ্গ করা হয়েছে।  
গীর্জ আমার সুল রিকোয়েস্টিকে বিভিন্ন করবেন, ধৰ্মবাদ।

Attach files by dragging & dropping, selecting or pasting them.

Allow edits by maintainers Create pull request

Remember, contributions to this repository should follow its code of conduct.

এবার **Create pull request** বাটনে ক্লিক করে পুল রিকোয়েস্ট তৈরি করে ফেলুন। ব্যাস হয়ে গেলো আপনার অন্য একটা প্রোজেক্টে পুল রিকোয়েস্ট তৈরিঃ

The screenshot shows a GitHub pull request page. At the top, there's a header with the repository name 'devsonket/devsonket.github.io' (Public), and buttons for 'Edit Pins', 'Watch (27)', and a profile icon. Below the header, there are navigation links: 'Code', 'Issues (29)', 'Pull requests (2)' (which is underlined in red), 'Discussions', 'Actions', 'Projects (2)', 'Security', and a settings gear icon.

The main title of the pull request is 'astrojs typo fix #925'. Below it, there's a message: 'zonayedpcadot... wants to merge 1 commit into devsonket:develop from zonayedpcadotcom:astro-typo-fix'. There are four tabs below the title: 'Conversation (0)', 'Commits (1)', 'Checks (0)', and 'Files changed (1)'. A comment from 'zonayedpcadot...' is visible, timestamped '1 minute ago'. The comment text is: 'এখানে Framework বানানটাকে বাংলায় "ফ্রেমওয়ার্ক" না লিখ "ফ্রেমওয়ার্ক" লেখা হয়েছে। এই পুল রিকোয়েস্টে এই বানানটাইকেই ফিল্ট্র করা হচ্ছে। শীর্জ আমার মূল রিকোয়েস্টটিকে রিভিউ করবেন, ধন্যবাদ।' The comment has a timestamp '21e2e12' at the bottom right. There's also a small profile icon next to the author's name.

এবার আমার দেখানোমতো করে ডেভসংকেতে অন্যকোথাও কন্ট্রিবিউট করতে পারেন কিনা সেটা চেষ্টা করুন। অথবা সেইম জিনিসটা নিয়ে প্র্যাকটিস করতে চাইলে <https://github.com/WithZonayed/mistake-devsonket.github.io> এখানে চেষ্টা করুন।

# এখনো শেষ হয়নি

তবে এখন আপনি জাস্ট পুল রিকোয়েস্ট তৈরি করেছেন, অর্থাৎ আপনি উক্ত রিপোজিটরির মেইটেইনারদের কাছে রিকোয়েস্ট করেছেন আপনার পরিবর্তনগুলো প্রোজেক্টে অ্যাড করার জন্য। এখন রিভিউয়াররা আপনার পরিবর্তন রিভিউ করে উপযুক্ত মনে হলে সেটা মার্জ করবে। আর যদি উপযুক্ত বা কিছু মডিফাই করার প্রয়োজন মনে করে তাহলে আপনাকে চেঙ্গ রিকোয়েস্ট দিতে পারে। আপনি জাস্ট লোকালি আপনার কার্ডিনেট সেইম ব্রাওওই পরিবর্তনগুলো করে কমিট করে গিটহাবে পুশ দিলেই অটোম্যাটিক পুল রিকোয়েস্ট আপডেট হয়ে যাবে। এর জন্য আবার শুরু থেকে সবকিছু করতে হবে না।

08

# সেলিব্রেট



ব্যাস এভাবেই সবকিছু ঠিকঠাক থাকলে আপনার পুল রিকোয়েস মার্জ হয়ে যাবে আর আপনিও আপনার পরিবর্তনগুলো/ফিল্টাগুলো প্রোজেক্টের মেইন রিপোজিটরি অথবা এখানে ডেভসংকেত এর ওয়েবসাইটে(এই কন্ট্রিভিউশনের ক্ষেত্রে [devsonket.com/astro-js](https://devsonket.com/astro-js) তে) দেখতে পাবেন! কংগ্রাচুলেশন

## Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক

Astro JS জাভাস্ক্রিপ্ট ফ্রেমওয়ার্ক এর চিলিংচি এবং একটি ট্যাক্টিক এইচটিএমএল প্রবৎ কম জাভাস্ক্রিপ্ট  
পিয় ফ্রেমওয়ার্ক

কন্ট্রিভিউটর

উইদ জুনামোদ

প্রোজেক্টে কন্ট্রিভিউট

এক পলকে গিট ও গিটহাব - ১১৮

# এক্সপ্রেস

# গিট

# গিট রিস্টের

ওয়াফি তার প্রোজেক্টে নতুন কিছু একটা ডেভেলপ করা যায় কিনা সে চিন্তায় অনেকগুলো কোড লিখে, অনেকগুলো ফাইল আপডেট করলো। কিন্তু একটা সময় পর তার মনে হল যে না আসলে সে যে কাজটা করতে চাচ্ছে সেটা ঠিক এভাবে হচ্ছে না। এদিকে সে কাজটি করতে গিয়ে ১০-১৫ টার মতো ফাইলে নতুন করে কোড লিখেছে বা পুরনো কোড মডিফাই করেছে। এখন সে যেহেতু আর এটা হচ্ছে না তাই পূর্বের অবস্থায় ফিরে যেতে চাচ্ছে।

আপনারা এমন সিচুয়েশন হলে কি করতেন? নিশ্চয়ই ১০-১৫ টা ফাইল এক এক করে ওপেন করে নতুন কোড রিমুভ করে, আপডেটগুলো আগের অবস্থায় নেওয়ার চেষ্টা করতেন? কিন্তু ১০-১৫ টা ফাইলের ক্ষেত্রে সেটা কতটুকু নির্ভুলভাবে করা সম্ভব, বা আদৌ কি সম্ভব হতে

পারে? এক্ষেত্রে সবচেয়ে সহজ উভর হল এটা আসলে সম্ভব না, আর জটিল উভর হল চেষ্টা করে দেখতে পারেন বাট সেটা ঠিকঠাক আগের অবস্থায় ফিরে যাবে তার কোন গ্যারান্টি নাই। আর ঠিক এখানেই আসে গিট রিস্টোর কমান্ডের ম্যাজিক।

`git restore` কমান্ডটি মূলত আপনাকে কোন ফাইল বা ডিরেক্টরির আগের অবস্থায়(শেষ কমিটের অবস্থায়) ফিরিয়ে নিতে সাহায্য করে। এটা মূলত লোকাল আনকমিটেড চেঞ্জেসগুলোকে আনডু(পুর্বের অবস্থায় নিতে সাহায্য) করতে, অথবা `git add` দিয়ে স্টেজিং এ অ্যাড করা চেঞ্জেসগুলোকে আনডু করতে ব্যবহার করা যায়। ধরেন আপনার অলরেভি কমিটেড একটা প্রোজেক্টে নতুন একটা ফিচার ডেভেলপ করার চেষ্টা করছিলেন। কিন্তু কিছুদুর কোড করে যাওয়ার পর আপনার মনে হলো যে না এগুলো আসলে কমিট করার মতো হয়নি। আপনি আবার আগের অবস্থায় ফিরে যেতে চান। তাহলে সহজভাবে এই কমান্ডটি ব্যবহার করবেনঃ

```
> git restore <file>
```

ধরুন আপনার ফাইলের নাম যদি হয়ে থাকে `new-update.html`, তাহলে আপনি এটাকে আগের অবস্থায় নিতে পারবেন এভাবেঃ

```
> git restore new-update.html
```

সেইমভাবে একটা পুরো ডিরেষ্টরিকে আগের অবস্থায় আনতে চাইলেঃ

```
> git restore <directory>
```

আর যদি চান সবগুলো চেঞ্জেসকে আগের অবস্থায় ফিরিয়ে আনতেঃ

```
> git restore .
```

চেঞ্জ করা কিছু যদি অলরেডি স্টেজিং এ নিয়ে থাকেন তাহলেও আপনি সেটাকে আগের অবস্থায় নিয়ে আসতে পারবেন শুধুমাত্র --staged ফ্ল্যাগ ব্যবহার করেইঃ

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

```
> git restore --staged <directory>
```

```
> git restore --staged .
```

ধরুন আমার একটা প্রোজেক্টের একটা ফাইল `git-restore.txt` নিচের অবস্থায় কমিটেড আছেঃ

```
Hello this is git restore command.
```

এখন আমি নতুন ফিচার হিসেবে কিছু একটা অ্যাড করলাম সেইম ফাইলেং

```
Hello this is git restore command.
```

```
I have added this new line to check git restore command
```

এখন আমি সিদ্ধান্ত নিলাম, নাহ! আমি নতুন ফিচার বা এখানে লেখাটা রাখবো না। আমি শেষ কমিটের অবস্থায় ফিরে যেতে চাই। তাহলে খুব সহজেই রিস্টোর কমান্ডটি এখানে ব্যবহার করতে পারবো এভাবেঃ

```
> git restore git-restore.txt
```

ব্যাস এখন যদি ফাইলটা দেখেন, দেখবেন এটা এটার শেষ কমিটের  
অবস্থায় ফিরে গেছে:

```
Hello this is git restore command.
```

# গিটি স্ট্যাশ

সাহির তার প্রোজেক্টের একটা নতুন ভ্রাঞ্চে নতুন কোন ফিচার নিয়ে কাজ করছে। ফিচারটি বেশ বড় এবং সময়সাপেক্ষ কাজ, তবে সাহিরের ইতিমধ্যে সেটার অর্ধেক কাজ প্রায় শেষ। তবে কাজগুলো এখনো কমিট করার মতো অবস্থায় নেই। এরমধ্যেই কাজ করতে করতে হঠাৎ সে ওয়াফি থেকে ম্যাসেজ পেল যে প্রোজেক্টে নতুন একটা জিনিস এসেছে অন্য একটা ভ্রাঞ্চে, সেটা চেক করে ফিডব্যাক জানাতে হবে। এখন সাহির তার অর্ধেক করা কাজ ফেলে কিভাবে ওয়াফির সেই জিনিস চেক করবে? এতো কষ্ট করে করা কাজ কি সে গিট রিস্টোর কমান্ড ব্যবহার করে ফেলে দিবে?

এটার উত্তর হচ্ছে অবশ্যই না, প্রোজেক্টে গিট থাকতে আমাদের কেনো এমন হ্যাসেলে পড়তে হবে? ঠিক এসব কাজের জন্যেই রয়েছে গিট

এর স্ট্যাশ কমান্ডটি। গিট স্ট্যাশ কমান্ডের সাহায্যে আপনি আপনার করা অর্ধেক কাজটা একপাশে ফেলে রেখে অন্যান্য কাজ করতে পারবেন। তারপর আপনার সেই অন্য কাজ শেষ হলে আবার সেই কাজগুলো খুব সহজেই আরেকটা কমান্ড দিয়ে ফিরে পেয়ে যাবেন। আপনার অর্ধেক করা কাজ একপাশে রেখে দেওয়ার জন্যঃ

> **git stash**

কমান্ডটি দেওয়ার সাথে সাথে আপনার নতুন আনকমিটেড কাজগুলো নাই হয়ে যাবে। তবে তাতে ভয় পাওয়ার কিছু নাই, এটা জাস্ট আপনাকে ব্রাঞ্চ সুইচ করে ওয়াফির কাজগুলো দেখার সুযোগ করে দিবে। আপনি খুব সহজেই যখন ঐদিকের কাজ শেষ হয়ে যাবে তখন আবার এই চেঞ্জেসগুলো সুস্থ-স্বাভাবিক অবস্থায় ফিরে পাবেন নিচের এই কমান্ডটি ব্যবহার করেঃ

> **git stash pop**

মনে রাখবেন এই পপ কমান্ডটি আপনার সর্বশেষ স্ট্যাশ করা কাজগুলোই ব্যাক করবে এবং স্ট্যাশ লিস্ট থেকেও এটাকে ক্লিয়ার করে দিবে। তবে স্ট্যাশে যেহেতু আপনি একাধিক চেঞ্জেস রাখতে

পারবেন, সেক্ষেত্রে পপ দিতে থাকলে সবার শেষে অ্যাড করা চেঞ্জেসগুলো প্রথম হিসেবে পর্যায়ক্রমে আসতে থাকবে। তবে আপনি যদি চান যে আপনি চেঞ্জেসগুলো ফিরিয়ে নিয়ে আসবেন, আবার স্ট্যাশেও রেখে দিবেন, সেক্ষেত্রে আপনি নিচের কমান্ডটি ব্যবহার করতে পারেন। এই কমান্ডটির সুবিধা হলো আপনি চেঞ্জেস অ্যাপ্লাই করার পরও আবার স্ট্যাশ থেকে এগুলোর অ্যাক্সেস পাবেনঃ

```
> git restore apply
```

আমরা জেনেছি স্ট্যাশে একাধিক চেঞ্জের রাখা যায়, আমরা চাইলে সে চেঞ্জেসগুলোর লিস্টও দেখতে পারবো এভাবেঃ

```
> git stash list
```

যদি খোঝাল করেন তাহলে দেখবেন প্রত্যেকটা আইটেমের আগে এখানে `stash@{n}`, এখানে `n` মানে নাম্বার দিয়ে মার্ক করা আছে। আপনি এটা ব্যবহার করেও `pop` অথবা `apply` করতে পারবেনঃ

```
> git stash pop stash@{3}  
> git stash apply stash@{1}
```

যদি স্ট্যাশ লিস্ট ক্লিয়ার করে ফেলতে চান তাহলে এই কমান্ডটি  
ব্যবহার করবেনঃ

> **git stash clear**

কোন স্পেসেফিক আইটেম স্ট্যাশ থেকে রিমুভ করতে চাইলেঃ

> **git stash drop stash@{n}**

তবে এখানে একটা বিষয় অবশ্যই খেয়াল রাখতে হবে, সেটা হচ্ছে  
স্ট্যাশ কমান্ডটি এভাবে ব্যবহার করলে আপনার অলরেভি গিটে ট্র্যাক  
করা(কমিটেড) ফাইল বা ডিরেষ্টরির চেঞ্জেসগুলোও স্ট্যাশে রাখবে।  
আপনি যদি আপনার শেষ কমিটের পর একেবারে নতুন কোন ফাইল  
অথবা ডিরেষ্টরি অ্যাড করে থাকেন তাহলে সেগুলো এভাবে স্ট্যাশে  
যাবে না, সেক্ষেত্রে আপনাকে একটা ফ্ল্যাগ ব্যবহার করে গিটকে বলে  
দিতে হবে যে আপনি নতুন ফাইল, ডিরেষ্টরিগুলোও স্ট্যাশে নিতে  
চাচ্ছেনঃ

> **git stash -u**

# গিট রিসেট

ওয়াফি একদিন কাজ করতে গিয়ে ভুলে এখনো পুরোপুরি রেডি না এমন কিছু কোড কমিট করে ফেলে। পরবর্তিতে সে এটা নিয়ে খুবই টেনশনে পরে যায় যেহেতু কমিট করে ফেলেছে। কিন্তু সাহির তখন তাকে বললো আরে আমরা তো গিট ব্যবহার করছি, সমস্যা কোথায়? এখানে এমন সিচুয়েশনও খুব সহজেই হ্যান্ডেল করা যায় `git reset` কমান্ডের সাহায্যে।

হ্যাঁ আর ঠিক এইধরনের সিচুয়েশনের জন্যেই আপনারা `git reset` কমান্ডটি ব্যবহার করে আপনাদের অলরেডি কমিট করা যেকোনো চেঞ্জেসগুলোকে পূর্বের অবস্থায় ফিরিয়ে আনতে পারবেন। আপনি যদি চান স্পেসিফিক কোন কমিটের পরবর্তি চেঞ্জেসগুলো আনড়ু করতে, তাহলে কমান্ডটা এভাবে দিবেনঃ

```
> git reset <commit_id>
```

কমিট আইডি পুরোটাও দিতে পারবেন অথবা git log --oneline এ যেমন শর্ট আইডি পাওয়া যায় সেটাও ব্যবহার করতে পারবেন।

এক্ষেত্রে আপনার উক্ত কমিটের পরবর্তি যে যে চেঞ্জেসগুলো ছিলো সেগুলো আনকমিটেড অবস্থায় চলে যাবে। তবে আপনি যদি চান যে উক্ত কমিটের পরবর্তি চেঞ্জেসগুলো একেবারেই চলে যাক তাহলে উপরোক্ত কমান্ডটি এভাবে দিতে হবেঃ

```
> git reset <commit_id> --hard
```

তবে মনে রাখবেন অলরেভি গিটহাবে বা আপনারা কোন রিমোট রিপোতে উক্ত কমিটসহ পুশ করে ফেলেন তাহলে এটা ব্যবহার করে কোন কমিট রিসেট করাটা রিকমেন্ড না, যেহেতু এক্ষেত্রে আপনার অন্যান্য কোলাবরেটররা কনফিউজড হয়ে যেতে পারে হঠাত একটা কমিট উধাও হয়ে গেলে। ঐরকম পরিস্থিতিতে git revert কমান্ডটি বেশী ইউজফুল।

# গিট রিভার্ট

git revert কমান্ডটিও অলরেভি আছে এমন কোন কমিটের চেঞ্জেসগুলোকে রিভার্ট বা বাদ দিতে ব্যবহার করা হয়। তবে এক্ষেত্রে রিভার্ট বা বাদটা দেওয়া হয় আরেকটা কমিটের সাহায্যে, এই কারণেই মূলত এটাকে রিভার্ট বলা হয়ঃ

```
> git revert <commit_id>
```

কমান্ডটি দেওয়ার পর কমিট ম্যাসেজ লিখার জন্য একটা প্রস্পট আসবে যেখানে আপনি চাইলে কাস্টম ম্যাসেজ দিতে পারেন অথবা ডিফল্টটা রেখেও :wq (write & quite) লিখে বেরিয়ে আসতে পারেন। গিট সেটআপ অনুযায়ী ভিন্ন হতে পারে, সেক্ষেত্রে ঐটাতে কমিট ম্যাসেজ মডিফাই করে বা ডিফল্টটা রেখে সেইভ করলেই হবে।

রিভার্ট হওয়ার পর যদি আপনি গিট লগ দেখেন তাহলে দেখবেন এখানে আরেকটা কমিট করা হয়েছে রিভার্ট করার জন্য।

git reset আর revert এর মধ্যে পার্থক্য মূলত এখানেই যে reset কমান্ড একটা কমিট পর্যন্ত থাকা চেঞ্জেসগুলো পর্যন্ত ফিরে যাওয়া যায় পরবর্তি সবগুলো কমিট বাদ দিয়ে। সেই সাথে এক্ষেত্রে নতুন কোন কমিট তৈরি হয় না। আর revert এর ক্ষেত্রে একটা কমিটের চেঞ্জেসগুলো সব বাদ দেওয়া হয় নতুন আরেকটা কমিট এর মাধ্যমে। সেক্ষেত্রে আপনার প্রোজেক্টে গিটহাবে বা রিমোট কোন রিপোতে থাকলে অন্যান্য কন্ট্রিভিউটরোও রিভার্টের কমিটটা দেখতে পাবে এবং কনফিউজড হবে না। এছাড়াও আপনি গিট লগ দেখলে দেখবেন যেখানে reset করা হলে কোন প্রমাণ বা লগ পাবেন না যেখানে revert করা হলে সেটার জন্য আরেকটা কমিট অর্থাৎ লগ দেখতে পাবেন।

# গিট রিবেস

সাহির প্রোজেক্টে নতুন একটি ফিচার নিয়ে কাজ করবে, তাই `main` ভাঞ্চি থেকে চেকআউট করে নতুন ফিচার ডেভেলপমেন্ট এর জন্য আরেকটা ভাঞ্চি `feature` ক্রিয়েট করলো। এখন সাহির তার নতুন `feature` ভাঞ্চি নতুন ফিচার নিয়ে কাজ করছে, অন্ন অন্ন করে কাজ করে সে তার কাজগুলোকে কমিটও করে যাচ্ছে। এরমধ্যে ওয়াফি আবার মেইন প্রোডাকশন `main` ভাঞ্চি আরো নতুন কিছু কাজ যুক্ত করেছে। এখন এদিকে সাহিরও চাচ্ছে তার `feature` ভাঞ্চিও যাতে `main` এর সেই নতুন কাজগুলো পাওয়া যায়। সেজন্য সে কি করতে পারে?

হ্যা এইরকম জিনিস কয়েকরকমভাবে হ্যান্ডেল করা যায়। সাহির চাইলে এখন `main` ভাঞ্চির সব চেঞ্চেসগুলো মার্জ করতে পারে তার

`feature` ব্রাঞ্চের সাথে। এখন সাহিত তার `feature` ব্রাঞ্চের সাথে `main` ব্রাঞ্চ মার্জ `git merge main` করলে নতুন একটা কমিট ত্রিয়েট হয়ে মাস্টার ব্রাঞ্চের নতুন আপডেটগুলো তার `feature` ব্রাঞ্চে চলে আসবে। এখানে খেয়াল করবেন যে এক্ষেত্রে নতুন একটা মার্জ কমিট তৈরি হবে। যেটা আবার `git log` দিলেও আপনারা দেখতে পাবেন। অনেকের কাছে এরকম অতিরিক্ত কমিট তৈরি হওয়াটা একটু আনন্দিন মনে হয়। অথবা এক্ষেত্রে সাহিত এর `feature` ব্রাঞ্চে এমন মার্জ কয়েকবার করতে থাকলে সেটার কমিট হিস্টোরিও এমন আনন্দিন হয়ে যাবে। এসব ক্ষেত্রে ভালো সমাধান হতে পারে গিট রিবেস।

গিট রিবেস করলে সাহিতের নতুন `feature` ব্রাঞ্চের বেইস চেঞ্জ হয়ে যাবে। সেক্ষেত্রে `main` এর সাথে রিবেসিং করলে `main` এর নতুন কমিটগুলো(চেঞ্জেসগুলো) তার `feature` ব্রাঞ্চে হ্রুণ চলে আসবে, আর সে `feature` ব্রাঞ্চে যেসব কাজ করেছে সেগুলোর কমিটগুলোও আবার রিঅ্যাপ্লাই করা হবে সেগুলোর উপর। এক্ষেত্রে মেইন কোডগুলোতে কোনোরূপ পরিবর্তন দেখতে না পেলেও গিট লগ চেক করলে দেখবেন যে কমিট হিস্টোরি আরো ক্লিন দেখা যাচ্ছে।

গিট রিবেস করতে হলে যে ব্রাঞ্চ রিবেস করতে চাচ্ছেন সেটাতে থাকা অবস্থায় যে ব্রাঞ্চের চেঞ্জেসগুলো আনতে চাচ্ছেন সে ব্রাঞ্চের নাম দিতে হবে। যেমন আমি `feature` ব্রাঞ্চে যদি `main` ব্রাঞ্চের

চেঞ্জেসগুলো আনতে চাই তাহলে আমি feature এ থাকাকালীন এরকম কমান্ড দিবোঃ

```
> git rebase main
```

ব্যাস, এবার এই কমান্ড দেওয়ার পড় গিট দেখবে যে আমার বর্তমান ব্রাঞ্চ feature এর সাথে প্রদত্ত ব্রাঞ্চ main এর মধ্যে ঠিক সর্বশেষ কোন কমিটে মিল রয়েছে। যে কমিটের সাথে সর্বশেষ মিল রয়েছে দুইটা ব্রাঞ্চের, feature ব্রাঞ্চের ক্ষেত্রে ঠিক এর পরের কমিটগুলোকে একপাশে রাখা হবে। তারপর main ব্রাঞ্চের এর পরের কমিটগুলোকে feature ব্রাঞ্চে আনা হবে। তারপর একপাশে রাখা feature ব্রাঞ্চের কমিটগুলোকে আবার সিরিয়ালি ত্রিগুলোর পর অ্যাপ্লাই করা হবে। আর ঠিক এভাবেই মার্জের চাইতে ক্লিনার গিট কমিটের হিস্টোরি পাওয়া যাবে। আপনারা git log কমান্ড ব্যবহার করেও দুইটার পার্থক্য দেখতে পাবেন।

তবে এই git rebase কমান্ডটি এরকম ইউজফুল হলেও কোন পাবলিক রিপোজিটরিতে থাকা ব্রাঞ্চে অথবা টিমের একাধিক পার্সন কাজ করছে এমন ব্রাঞ্চে ব্যবহার না করাই বেটার। অথবা যদি ব্যবহার করেনও তাহলে সেটা অবশ্যই সবাইকে ইনফর্ম করে করতে হবে, অন্যথায় এখানে বেশ বড় ঝামেলা হয়ে যেতে পারে। গিট রিভেস

যেহেতু এক্সিটিং কমিটের হিস্টেরি চেঙ্গ করে ফেলে, এমনকি কমিটের হ্যাশও(কমিট আইডি) চেঙ্গ করে ফেলে, সেইম ব্রাঞ্চে কাজ করা আরেকজন তখন আপনি রিবেস করলে সেগুলোর চেঞ্জেসগুলো আর পুল করতে পারবে না নরমালভাবে। অতএব এই কমান্ড ব্যবহার করার পূর্বে ভালোভাবে বুঝে নেওয়া জরুরী আপনি ঠিক কোথায় ব্যবহার করছেন আর এটা করার কারণে অন্যান্য কোলাবরেটরদের কোন সমস্য হবে কিনা ইত্যাদি ইত্যাদি। তবে শুধুমাত্র আপনি কাজ করছেন বা আপনার লোকালে আছে এমন ব্রাঞ্চে এই কমান্ড ব্যবহার করতে কোন বাঁধা নেই। খালি রিমোট রিপোতে থাকা কোন এক্সিটিং কমিটকে উলটপালট না করলেই হচ্ছে।

# গিট স্কোয়াশিং

ওয়াফি প্রোজেক্টে নতুন একটি ফিচার নিয়ে কাজ করার চিন্তাবনা করে, তাই সে তাদের প্রোডাকশন main ভাঞ্চ থেকে চেকআউট করে নতুন আরেকটি ভাঞ্চ new-feature তৈরি করলো। এখন সে এই নতুন ভাঞ্চে তার ডেভেলপমেন্ট শুরু করলো। সে ডেভেলপমেন্ট শেষ করার পর সেটা সাহিতের সাথে শেয়ার করে সিদ্ধান্ত নিলো তারা এটা তাদের মেইন প্রোডাকশন main ভাঞ্চে মার্জ করবে। এখন তারা খেয়াল করলো তার এই new-feature ভাঞ্চে অনেকগুলো কমিট তৈরি করা হয়েছে ডেভেলপমেন্ট এর সময়, অথচ ফিচারটা খুবই ছোটো একটা ফিচার। কমিট হিস্টোরিতে এতগুলো ছোটো ছোটো কমিট থাকলে সেটা একটু আনাক্লিন দেখা যেতে পারে। এই অবস্থায় বেটার হয় যদি new-feature ভাঞ্চের সব আপডেট

জাস্ট একটা কমিটের মাধ্যমে মেইন main ভাবে আনা যায়। হ্যাঁ!  
ঠিক সেইম কাজটাই সম্ভব গিট স্কোয়াশিং এর সাহায্যে।

গিট স্কোয়াশিং এর ক্ষেত্রে আপনি আপনার যে ভাবেন(এখানে main ধরলাম) সেখানে গিয়ে আপনি যে ভাবের চেঞ্জেসগুলো মার্জ করবেন সেটা উল্লেখ করে কমান্ড দিবে, তবে স্কোয়াশ মানে সবগুলো কমিটকে একসাথে করার জন্য এক্সট্রা একটা ফ্ল্যাগ --squash ব্যবহার করতে হবে এভাবেঃ

```
> git merge new-feature --squash
```

ব্যাস এবার সবগুলো চেঞ্জ আপনার কারেন্ট ভাবে চলে আসবে, তবে সেগুলো স্টেজড করা অবস্থায় পাবেন। আপনি এখন জাস্ট আরেকটা কমিট করে সবগুলো চেঞ্জেস জাস্ট একটা কমিটের মাধ্যমে আপনার ভাবে যুক্ত করে ফেলতে পারবেনঃ

```
> git commit -m "new feature introduced"
```

ব্যাস এভাবেই হয়ে গেলো আরো ক্লিনার গিট কমিট হিস্টোরি।

অন্যান্য

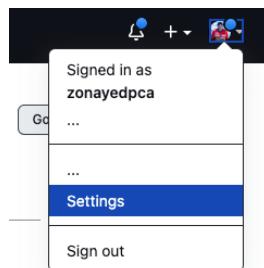
# গিটহাব ব্যবহার করব না

হ্যাঁ এটা আপনার নিজের ইচ্ছা, আপনি চাইলে গিটল্যাব (gitlab.com) বা বিটবাকেট (bitbucket.org) ব্যবহার করতে পারেন সেটা সম্পূর্ণ আপনার নিজের ইচ্ছা বা প্রয়োজনের উপর নির্ভর করে। মেইন ফাংশনালিটি একই, কারণ এখানে মেইন কাজটা করবে গিট। গিটের কমান্ড সবই এক থাকবে। শুধুমাত্র আপনার হোস্টিং প্রোভাইডার অনুযায়ী ইউআই একটু ডিফারেন্ট হতে পারে, রিমোট ইউআরএল আপনার হোস্টিং প্রোভাইডারের দেওয়া ইউআরএল অনুযায়ী অ্যাড করে নিবেন। বাট মেইন আইডিয়া/কন্সেপ্ট সবারই মোটামুটি এক। তো আপনার নিজের ইচ্ছা মত আপনি যে কোনো একটা ব্যবহার করতে পারবেন।

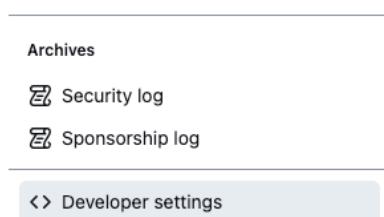
# গিটহাবে SSH ব্যবহার না করা

আমরা শুরুর দিকে গিটহাবে SSH কী সেটআপ করে নিয়েছিলাম সহজে সিকিউরলি গিটহাবের বিভিন্ন রিপোতে কাজ করার জন্য। কিন্তু যদি এরকম কোন সিচুয়েশন আসে যে আমাদের কোন কারণে টেম্পোরারি অথবা খুবই লিমিটেড অ্যাক্সেস দেওয়া লাগে কোন একটা ম্যাশিনে বা সিস্টেমে, যেমনং আপনি জাস্ট ঐ ম্যাশিন থেকে আপনার বিশেষ একটা বা একাধিক প্রোজেক্ট বা রিপোতেই বিশেষ কিছু কাজ যেমনং জাস্ট রিপো রিড করতে পারা বা ইত্যাদি ইত্যাদি করতে চান তাহলে আপনার জন্য গিটহাবে SSH কী ব্যবহার করা ছাড়া আরেকটা অপশনও আছে।

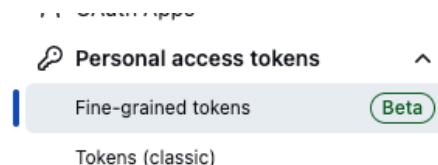
সেক্ষেত্রে প্রথমে আপনাকে আপনার গিটহাব অ্যাকাউন্ট থেকে সেটিংস এ যেতে হবে।



তারপর এবার ডানপাশে একদম নিচে একটা মেনু পাবেন  
Developer settings নামেং



এখান থেকে এবার আপনি অনেকগুলো অপশন পাবেন ডানপাশের মেনুতে, তারমধ্যে আমাদের ফোকাস থাকবে Personal access tokens মেনুতে। এটার আভারে আরো দুইটা সাব-মেনু পাবেন Fine-grained tokens আর Tokens(classic)। যদিও ক্ল্যাসিক মেথড ব্যবহার করেও সেইম কাজটা করতে পারবেন, তবে ফাইন গ্রেইনেড(যেরকম নাম, সেরকম কাজ) আরো বেশী কন্ট্রোল পাবেন কিভাবে আর কতটুকু লিমিট দিতে চান আপনি আপনার টোকেনে সেসব বিষয়ের উপর।



তারপর এখান থেকে বামপাশের Generate new token বাটনে  
ক্লিক করবেনঃ

### Fine-grained personal access tokens (Beta)

[Generate new token](#)

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

তারপরে এখানে অনেক অপশন দেখে একটু কনফিউশন তৈরি হতে  
পারে, বাট আস্তে আস্তে পডলেই কোনটার কি কাজ সব বুঝতে  
পারবেন।

### New fine-grained personal access token (Beta)

Create a fine-grained, repository-scoped token suitable for personal API use and for using Git over HTTPS.

Token name \*

আপনার টোকেনের নাম 

A unique name for this token. May be visible to resource owners.

Expiration \*

30 days  The token will expire on Tue, Apr 11 2023

Description

একটা বৰ্ণনা জাও টোকেনটা কি কাজে ইউজ কৰছেন সেটা মনে রাখার জন্য

What is this token for?

Resource owner

 zonayedpcadotcom 

টোকেন সম্পর্কে বিস্তারিত

প্রথম ফিল্ডে আপনার টোকেনের নাম(Token name), তারপর  
টোকেনটা কতদিন পর এক্সপায়ার হয়ে যাবে সেটা সিলেক্ট কৰার  
অপশন(Expiration)। তারপর টোকেনটা কি কাজে ব্যবহার

করছেন সেটার ব্যাপারে একটু বর্ণনা(Description)। তারপর আপনার কোন রিসোর্স থেকে এটাকে অ্যাক্সেস দিতে চাচ্ছেন সেগুলো সিলেক্ট করার অপশন(Resource Owner)। এখানে আপনার ইউজারনেমই দেখাবে, আর অর্গানাইজেশন থাকলে সেগুলোর লিস্টও দেখাবে।

#### Repository access

Public Repositories (read-only)

All repositories

This applies to all current *and* future repositories owned by the resource owner.  
Also includes public repositories (read-only).

Only select repositories

Select at least one repository. Max 50 repositories.  
Also includes public repositories (read-only).

Select repositories ▾

Selected 1 repository.

 zonayedpcadotcom/Hello-World

x

রিপোজিটরি অ্যাক্সেস

তারপর এখানে আপনি কোন ধরনের অ্যাক্সেস দিতে চাচ্ছেন সেটার বিস্তারিত। Public Repositories সিলেক্ট করলে সব পাবলিক রিপোর রিড ওয়ানলি অ্যাক্সেস দেওয়া হবে, All repositories বা Only select repositories সিলেক্ট করলে সব/স্পেসেফিক রিপোজিটরির পারমিশন কোনগুলো দিতে চান সেটা নিচে দেখাবে। আমরা আমাদের অ্যাকাউন্টে থাকা Hello-World নামে একটা রিপোর অ্যাক্সেস দিয়েছি, এখন এটার পারমিশন কি কি থাকবে

সেগুলোও নিচে থেকে স্পেসেফিকভাবে সিলেক্ট করার অপশন  
থাকবেং

## Permissions

Read our [permissions documentation](#) for information about specific permissions.

**Repository permissions** (2 Selected) 

Repository permissions permit access to repositories and related resources.

<b>Commit statuses</b> ⓘ Commit statuses.	<button>Access: No access ▾</button>
<b>Contents</b> ⓘ Repository contents, commits, branches, downloads, releases, and merges.	<button>Access: Read and write ▾</button>
<b>Merge queues</b> ⓘ Manage a repository's merge queues	<button>Access: No access ▾</button>
<b>Metadata</b> ⓘ <span style="border: 1px solid orange; padding: 2px;">mandatory</span> Search repositories, list collaborators, and access repository metadata.	<button>Access: Read-only ▾</button>
<b>Workflows</b> ⓘ Update GitHub Action workflow files.	<button>Access: No access ▾</button>

**Account permissions**   
User permissions permit access to resources under your personal GitHub account.

## রিপোজিটরি পারমিশন

আমরা এখানে Repository permissions এ জাস্ট উপরে  
সিলেক্টকৃত রিপোতে কন্টেন্টস(Contents) Read and write  
অ্যাক্সেস দিয়েছি(এটা সেট করলে আরেকটা ফিল্ড(Metadata)  
অটোই Read-only অ্যাক্সেস পেয়ে যায় যেহেতু এটাও ম্যান্ডেটরি)।  
আর এই পারমিশনটাই আমাদেরকে উক্ত রিপো থেকে পুল পুশ করতে

পারমিশন দিবে। তবে আপনি চাইলে অন্যান্য অপশনগুলো আপনার চাহিদামতো ঘাটাই করে টোকেন তৈরি করতে পারেন। এখানে আপনাকে একদম ফাইন গ্রেইন্ড কন্ট্রোল দেওয়া হবে যাতে আপনার টোকেন মিস-ইউজ না হতে পারে। বাট আমি দেখানোর সুবিধার্থে জাস্ট এই পারমিশনগুলোই সিলেষ্ট করেই নিচে Generate token বাটনে ক্লিক করে টোকেন ত্রিয়েট করে নিবঃ

### Overview

2 permissions for 1 of your repositories >  
0 Account permissions >

This token will expire April 11, 2023.

[Generate token](#) [Cancel](#)

This token will be ready for use immediately.

এবার পরের পেজে টোকেনটা দেখানো হবে, এটা আপনি কপি করে কোথাও নিরাপদে রেখে দিতে পারবেন। তবে মনে রাখবেন গিটহাব শুধুমাত্র এই একবারই এই টোকেন আপনাকে দেখাবে, এটা আর ভবিষ্যতে কেউ চাইলেও গিটহাব থেকে আর দেখতে পাবে নাঃ

These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS.

Make sure to copy your personal access token now as you will not be able to see this again. Never used [Delete](#)

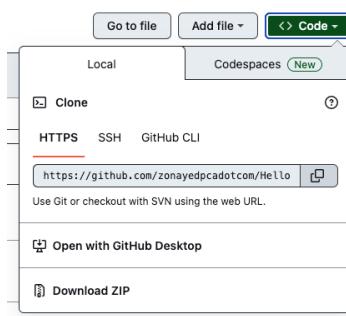
github\_pat\_11ANH3BAY0CCDDdf5IHYTp\_aVUJT6HhS0BVi8TPw [Copy](#)

Expires on Tue, Apr 11 2023. [Copy to clipboard](#)

এই টোকেনটা শুধুমাত্র এই একবারই দেখতে পাবেন

এবার ব্যাস আপনার টোকেন রেডি, এবার আপনি এই টোকেন ব্যবহার করে আপনার পারমিশন দেওয়া রিপোতে SSH সেটআপ করা ছাড়াই সরাসরি পুল পুশ করতে পারবেন। তবে এখানে আরেকটা কথা আছে, সেটা হচ্ছে এই রিপোটার SSH লিংক এর পরিবর্তে এটার ক্ষেত্রে আপনাকে HTTPS লিংক ব্যবহার করতে হবে। আর পুশ করার সময় পাসওয়ার্ডের পরিবর্তে জেনারেট করা টোকেনটা দিতে হবে।

ক্লোন করার সময় এখান থেকে SSH সিলেক্ট না করে HTTPS সিলেক্ট করে ক্লোন করতে হবে যেহেতু আমরা আর SSH ব্যবহার করছি নাঃ



ব্যাস এভাবেই আপনি খুব সহজে অথবা বলা যায় খুব সাবধানতা অবলম্বন করে টেম্পোরারি অথবা খুবই লিমিটেড অ্যাক্সেস দরকার এরকম জায়গা থেকেও আপনি আপনার রিপো বা প্রোজেক্ট এর অ্যাক্সেস পাবেন।

# স্টুডেন্টদের জন্য গিটহাবের অফার

গিটহাব স্টুডেন্টদের জন্য প্রিমিয়াম ফিচারগুলা ছীতে দেয়। এর জন্যে গিটহাবের স্টুডেন্ট ডেভেলপার প্যাক নিতে পারবেন আপনার ইউনিভার্সিটির আইডি কার্ড বা ইউনিভার্সিটি থেকে দেওয়া ইমেইল ব্যবহার করে। আপনাকে শুধু এই লিঙ্কে[ <https://education.github.com/pack> ] গিয়ে Get Your Pack এ ক্লিক করতে হবে। তারপর আপনার আইডি কার্ডের স্ক্যান কপি অথবা ইমেইল আর দুই একটা অন্যান্য ডিটেইলস সাথে ইন্সটিউটের নাম দিয়ে অ্যাপ্লাই করে দিতে হবে। গিটহাব ভেরিফিকেশন প্রসেস কমপ্লিট করে আপনাকে প্রিমিয়াম গিটহাব ফিচারগুলোর অ্যাক্সেস দিবে দুই বছরের জন্য। আনলিমিটেড প্রাইভেট রিপোজিটরি ক্রিয়েট করতে পারবেন। দুই বছর পর আবার অ্যাপ্লাই

করে আবারো সেইমভাবে দুইবছর পাবেন, মানে যতক্ষণ পর্যন্ত স্টুডেন্ট  
আছেন ততক্ষণ পর্যন্ত পাবেন।

# আরো কিছু

১। প্রোজেক্টে কাউকে কন্ট্রিবিউটর হিসাবে অ্যাড করাঃ আপনার গিটহাবের প্রোজেক্টের পেজে গিয়ে উপরে দেখুন Settings নামে একটা ট্যাব আছে। এখানে লিঙ্ক করলে পরের পেজে একটা সাইডবার পাবেন। এখানে Collaborators নামে একটা মেনু আছে। এখানে লিঙ্ক করলে পরের পেজে Collaborator অ্যাড করার জন্যে ফর্ম আসবে। যাকে অ্যাড করতে চান তার ইউজারনেম দিলে তাকে দেখাবে ড্রপডাউনে। এখান থেকে সিলেক্ট করে দিলেই Collaborator হিসেবে অ্যাড হয়ে যাবে। এখানে যেহেতু অনেক গুরুত্বপূর্ণ কাজ করতেছেন তাই কোনো কোনো স্টেপে পাসওয়ার্ড দিতে হতে পারে।

২। রিভমি ফাইল অ্যাড করাঃ মাঝে মধ্যে গিটহাবের প্রোজেক্ট পেজে দেখবেন নিচে প্রোজেক্টের বিস্তারিত তথ্য দেওয়া থাকে। এজন্যে আপনার প্রোজেক্টের রুটে **README.md** নামে একটা ফাইল অ্যাড করতে হবে। আর এর ভিতরে [মার্কডাউন সিনট্যাক্স](https://with.zonayed.me/post/md-at-a-glance)(with.zonayed.me/post/md-at-a-glance) ব্যবহার করে ভিতরে লিখতে হবে। গিটহাব অটোম্যাটিক্যালি এখানকার কন্টেন্টগুলো বিস্তারিত আকারে প্রোজেক্ট পেজে দেখাবে।

৩। কোনো ডিরেক্টরি বা ফাইল ইগনোর করাঃ মাঝেমধ্যে আমাদের প্রোজেক্টে এমন কোনো ফাইল বা ডিরেক্টরি থাকতে পারে যেটা গিট ট্র্যাক করুক আমরা সেটা চাই না। সে জন্য .gitignore নামে একটা ফাইল বানাতে হবে রুট ডিরেক্টরিতে আর ভিতরে কোন কোন ফাইল বা ডিরেক্টরি ইগ্নোর করতে হবে সেগুলো মেনশন করে দিতে হবে। ভালো একটা উদাহরণ node\_modules ডিরেক্টরি ইগ্নোর করা। গিটহাবে প্রোজেক্ট সেটাপ করার সময়ও লক্ষ্য করলে নিচে দেখবেন এটা সেটাপ করার একটা অপশন থাকে। আমি জাস্ট আইডিয়া দিলাম, বাকিটা গুগুল করলে পেয়ে যাবেন।

# শেষ কথা

আমি এই সিরিজে যা দেখালাম এগুলোর বাইরেও আরো অনেক কাজ রয়েছে গিট এবং গিটহাবের। অনেক কিছু জানার এবং শেখার মতো আছে। তবে মেইন বেস এগুলোই। আপনি এই ব্যাসিকগুলো জানলে পরে বাকিগুলো গুগল করে, অথবা কাজ করতে করতেও শিখে ফেলতে পারবেন। আসলে গিট ও গিটহাবের মেইন ধারনাটা ধরতে পারাই হলো আমার এই বইয়ের আসল উদ্দেশ্য।

আর অবশ্যই বইয়ের ব্যাপারে যেকোনো ফিডব্যাক, ভুল-ভাস্তি, মতামত, জিজ্ঞাসা বা যেকোনোকিছু জানার থাকলে আমাকে ইমেইল করতে পারেনঃ [contact@zonayed.me](mailto:contact@zonayed.me) তে অথবা ওয়েবসাইট থেকে([with.zonayed.me/contact](http://with.zonayed.me/contact)) সরাসরি কন্ট্যাক্ট করতে পারেন, অসংখ্য অসংখ্য ধন্যবাদ!

# বিশেষ ধন্যবাদ

১। শাহাদাত শিমুল (shahadat.pro)

২। প্রোগ্রামিং পাতা ([link.zonayed.me/PPataBN](http://link.zonayed.me/PPataBN))

# কেউ তো বাংলায় ই-বুকের কথা বলেনি!

বাংলাতে ই-বুক খুব বেশী একটা দেখা যায় না। এমনিতে সচরাচর হার্ডকপিওয়ালা বইয়ের স্ক্যান কপির ই-বুক (লিগ্যালি হার্ডক অথবা ইলিগ্যালি) বেশ দেখা যায়। তারপরেও আমরা আসলে ই-বুকের চাইতে ফিজিক্যাল, হার্ড কপির বই-ই বেশী পছন্দ করি।

কিন্তু বর্তমানে আসলে সময়ের পরিবর্তন হয়েছে। ই-বুক ক্যারি বা স্টোর করতে যেমন এক্স্ট্রা কোন হ্যাসেল নেই, ঠিক তেমনভাবে আমাদের জন্য জ্ঞানের সাগরও নতুনভাবে উন্মুক্ত করে দিয়েছে। ই-বুক আপনি যেকোনো সময়, যেকোনো জায়গায় জাস্ট আপনার সাথে স্ক্রিনওয়ালা একটি ডিভাইস, বা স্মার্টফোন থাকলেই স্টো ইউজ করে সহজেই পড়া শুরু করতে পারবেন। আমাদের মতো ট্রাফিক জ্যাম প্রধান দেশে তো ই-বুক আপনার ট্রাফিকের পেছনে নষ্ট হওয়া সময়গুলোকে নিজের জ্ঞান সমৃদ্ধ করার কাজে বেশ ভালোভাবে লাগতে পারে। আর একটা বিষয় উল্লেখ না করলেই নয় যে কোয়ালিটি গ্রাফিক্স, ইমেজ, লার্নিং ম্যাটেরিয়াল সমৃদ্ধ একটা বইয়ের চাইতে ই-বুকের পেছনে খরচ অনেক কম হয়ে থাকে।

সেই লক্ষ্যেই আমি উইদ জুনায়েদ এর পক্ষ থেকে ই-বুক লঞ্চ করার পারিকল্পনা হাতে নিয়েছি। প্রথমে তাই সবার কাজে আসতে পারে এমন একটা টপিকের উপরই এই বইটি রিলিজ করেছি। এক্ষেত্রে আপনার ফিডব্যাক অবশ্যই আমার জন্য অনেক মূল্যবান হবে, ধন্যবাদ।



উইদ জুনায়েদ একটি লার্নিং প্ল্যাটফর্ম, এখানে আমাদের প্রধান ফোকাস হচ্ছে ওয়েব, মোবাইল, ডেস্কটপ অ্যাপ ডেভেলপমেন্ট থেকে শুরু করে কম্পিউটার সাইন্স, প্রোগ্রামিং ল্যাঙ্গুয়েজসহ বিভিন্ন রেলিভেন্ট টপিকের উপর বিভিন্ন ফরম্যাটে (ব্লগপোস্ট, ভিডিও টিউটোরিয়াল, বই ইত্যাদি) লার্নিং কন্টেন্ট প্রোভাইড করা।

## বইটি বিক্রয়ের জন্যে নয়!

এই বইটি সম্পূর্ণ বিনামূল্যে উইদ জুনায়েদ এর ওয়েবসাইট [with.zonayed.me](http://with.zonayed.me) তে পাওয়া যাবে, কোনোভাবেই বিক্রয়ের জন্য নয়। নিজের জন্য ছাড়া কমার্শিয়াল প্রিন্ট করা যাবে না।